# Computer Networks

Quanlong Li

教育部高等教育司推荐
国外优秀信息科学与技术系列教学用书

计算机网络
——自顶向下方法
（第5版　影印版）

COMPUTER NETWORKING
A Top-Down Approach
(Fifth Edition)

James F. Kurose
Keith W. Ross

高等教育出版社

# Chapter 5: Link layer

*our goals:*

- understand principles behind link layer services:
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
  - local area networks: Ethernet, VLANs
- instantiation, implementation of various link layer technologies

# Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs
- addressing, ARP
- Ethernet
- switches
- VLANS

5.5 link virtualization: MPLS

5.6 data center networking

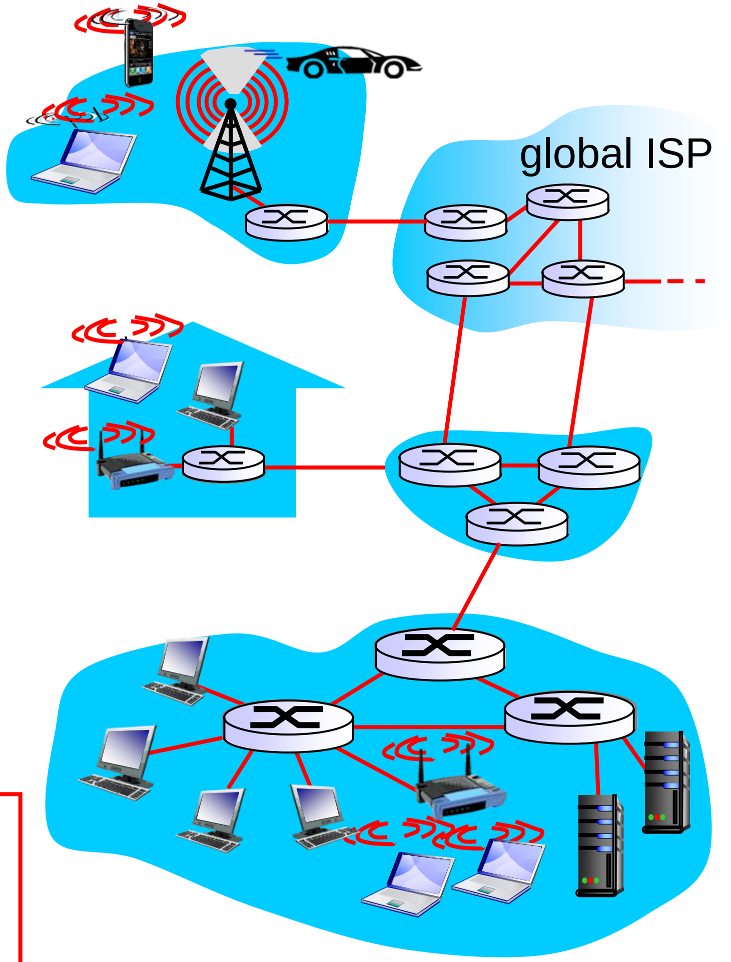
5.7 a day in the life of a web request

# Link layer: introduction

*terminology:*

- hosts and routers: nodes
- communication channels that connect adjacent nodes along communication path: links
  - wired links
  - wireless links
  - LANs
- layer-2 packet: frame, encapsulates datagram

*data-link layer* has responsibility of transferring datagram from one node to *physically adjacent* node over a link



# Link layer: context

- datagram transferred by different link protocols over different links:
  - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
- each link protocol provides different services
  - e.g., may or may not provide rdt over link

*transportation analogy:*

- trip from Princeton to Lausanne
  - limo: Princeton to JFK
  - plane: JFK to Geneva
  - train: Geneva to Lausanne
- tourist = datagram
- transport segment = communication link
- transportation mode = link layer protocol
- travel agent = routing algorithm

# Link layer services

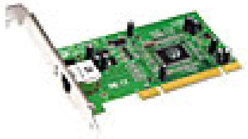
- *framing, link access:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses used in frame headers to identify source, dest
    - different from IP address!
- *reliable delivery between adjacent nodes*
  - we learned how to do this already (chapter 3)!
  - seldom used on low bit-error link (fiber, some twisted pair)
  - wireless links: high error rates
    - *Q:* why both link-level and end-end reliability?

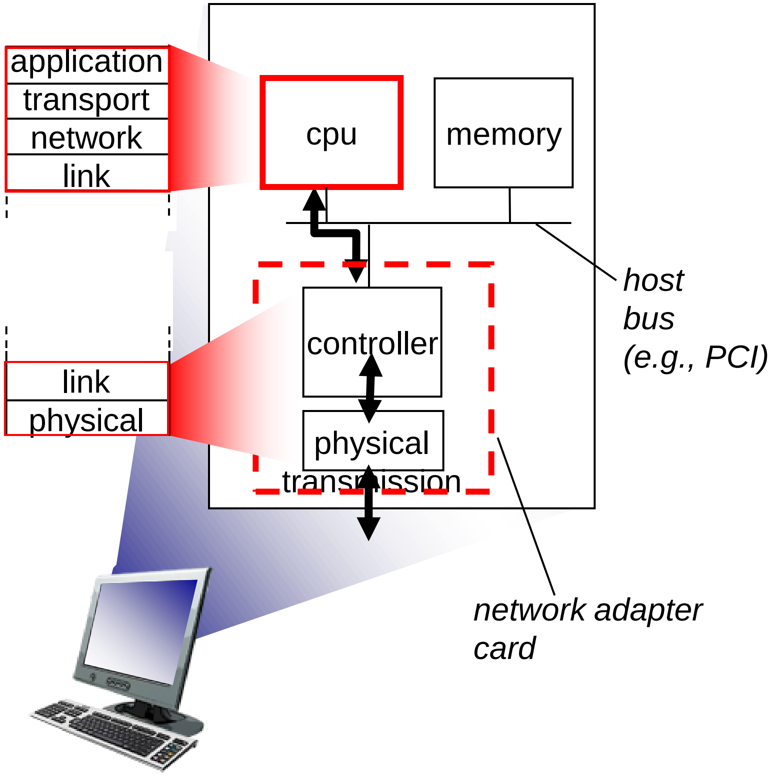
# Link layer services (more)

- *flow control:*
  - pacing between adjacent sending and receiving nodes
- *error detection:*
  - errors caused by signal attenuation, noise.
  - receiver detects presence of errors:
    - signals sender for retransmission or drops frame
- *error correction:*
  - receiver identifies *and corrects* bit error(s) without resorting to retransmission
- *half-duplex and full-duplex*
  - with half duplex, nodes at both ends of link can transmit, but not at same time

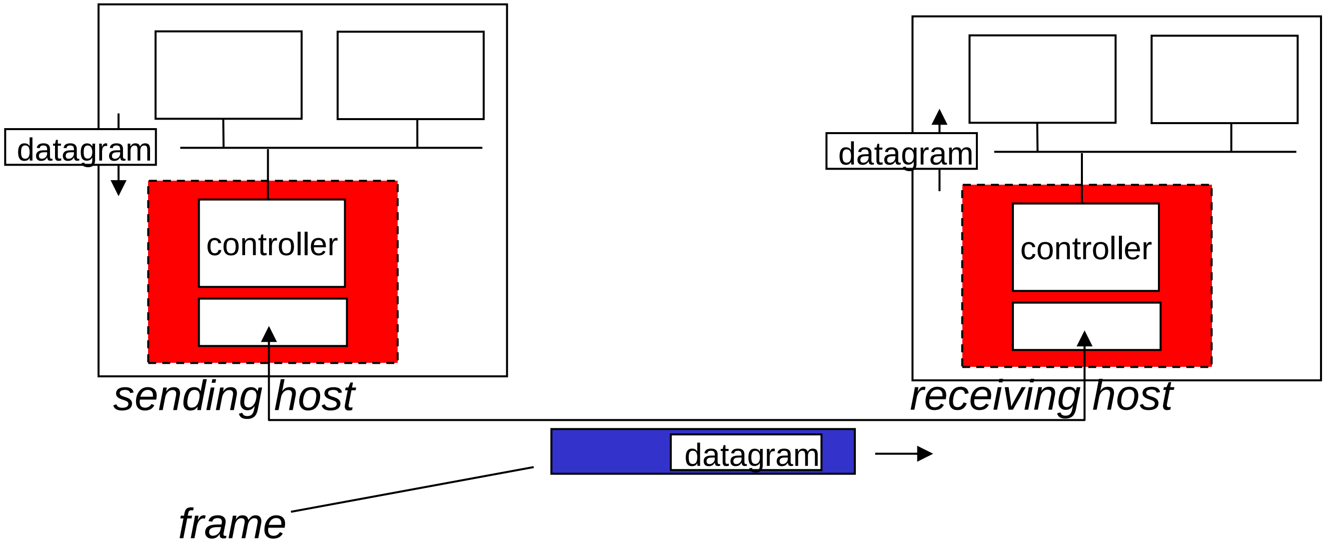
# Where is the link layer implemented?

- in each and every host
- link layer implemented in "adaptor" (aka *network interface card* NIC) or on a chip
  - Ethernet card, 802.11 card; Ethernet chipset
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware





# Adaptors communicating



- sending side:
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- receiving side
  - looks for errors, rdt, flow control, etc
  - extracts datagram, passes to upper layer at receiving side

# Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

5.5 link virtualization: MPLS

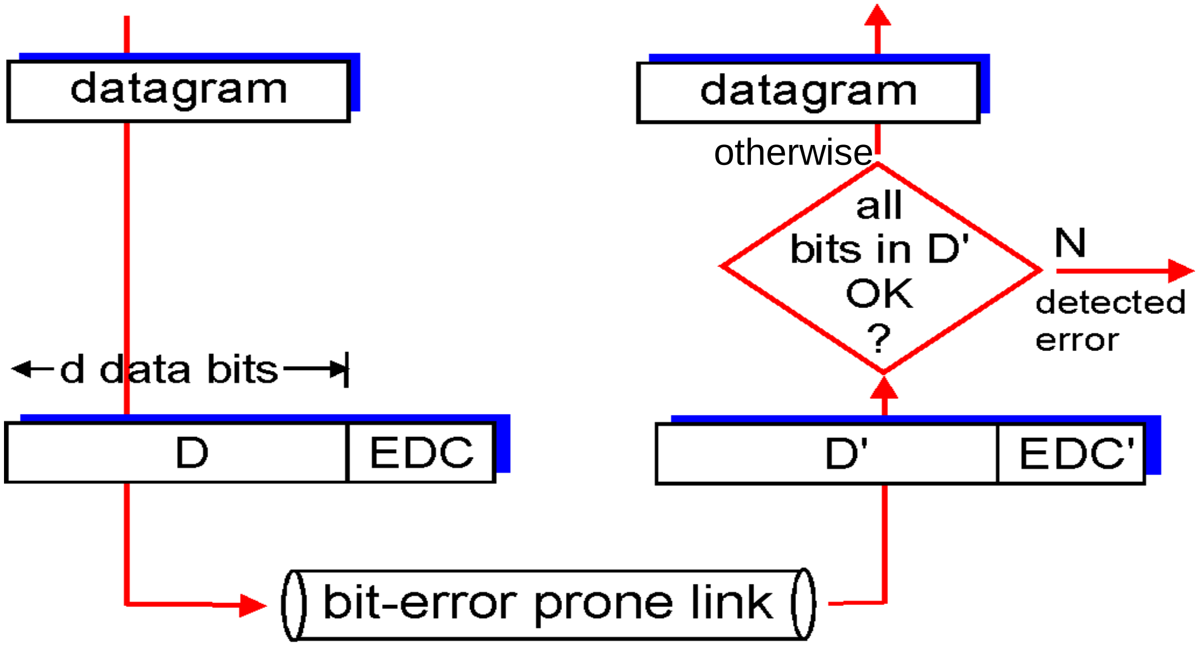
5.6 data center networking

5.7 a day in the life of a web request

# Error detection

EDC= Error Detection and Correction bits (redundancy)
D = Data protected by error checking, may include header fields

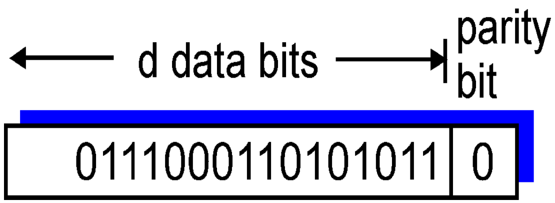
- Error detection not 100% reliable!
  - protocol may miss some errors, but rarely
  - larger EDC field yields better detection and correction



# Parity checking

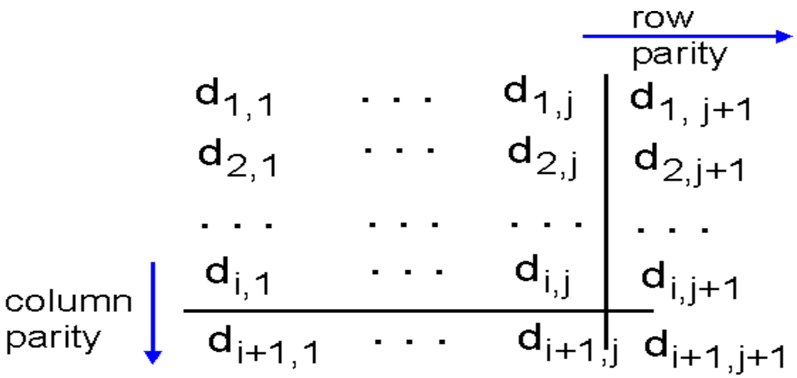
*single bit parity:*

- detect odd bits errors

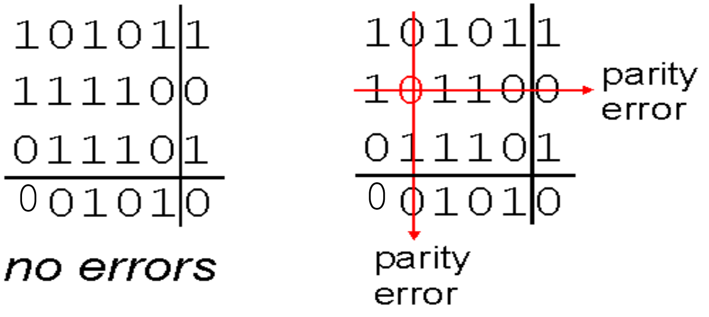


*two-dimensional bit parity:*

- detect and correct single bit errors



$$\begin{array}{cccc|c} d_{1,1} & \cdots & d_{1,j} & & d_{1,\,j+1} \\ d_{2,1} & \cdots & d_{2,j} & & d_{2,j+1} \\ \cdots & \cdots & \cdots & & \cdots \\ d_{i,1} & \cdots & d_{i,j} & & d_{i,j+1} \\ \hline d_{i+1,1} & \cdots & d_{i+1,j} & & d_{i+1,j+1} \end{array}$$



```
10101|1
11110|0
01110|1
------
00101|0
```

*no errors*

```
10101|1
10110|0
01110|1
------
00101|0
```

*correctable single bit error*

# Internet checksum (review)

*goal:* detect "errors" (e.g., flipped bits) in transmitted packet (note: used at transport layer *only*)

*sender:*

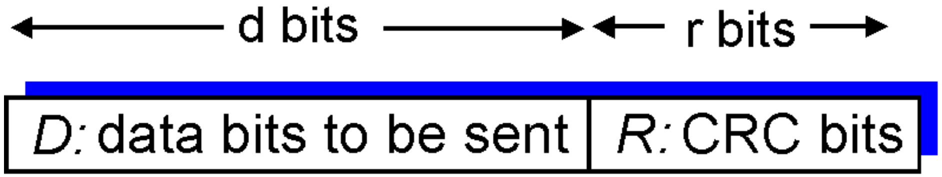
- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

*receiver:*

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?*

# Cyclic redundancy check

- more powerful error-detection coding
- view data bits, D, as a binary number
- choose r+1 bit pattern (generator), G
- goal: choose r CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
  - can detect all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi, ATM)



*bit pattern*

$$D * 2^r \quad \text{XOR} \quad R$$

*mathematical formula*

# CRC example

want:

$D \cdot 2^r$ XOR $R = nG$

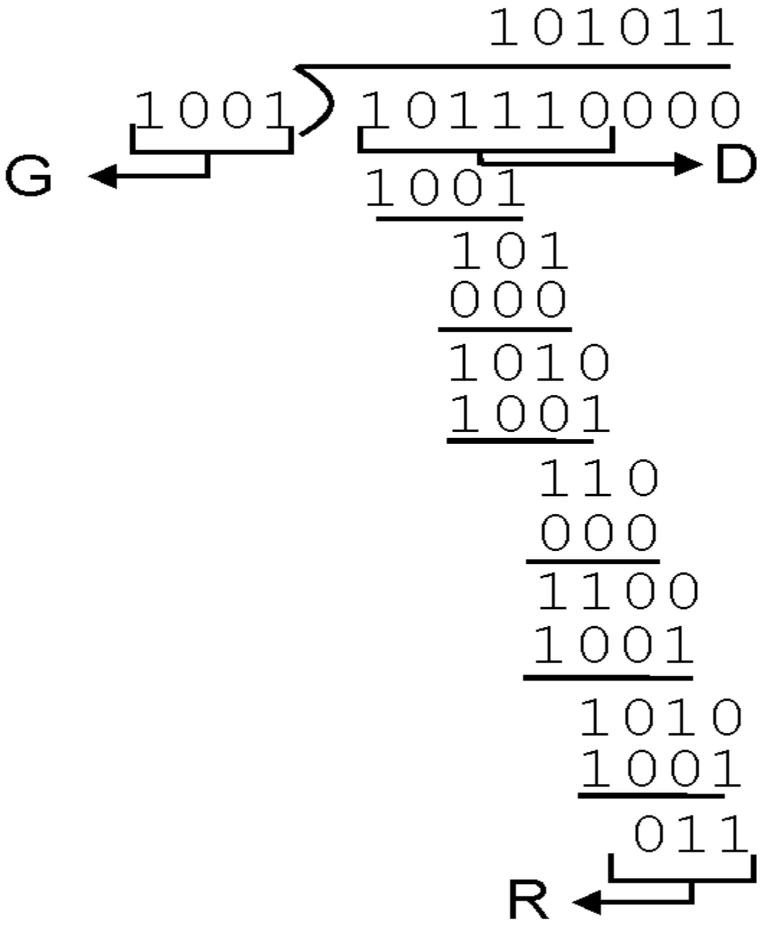
*equivalently:*

$D \cdot 2^r = nG$ XOR $R$

*equivalently:*

if we divide $D \cdot 2^r$ by G, want remainder R to satisfy:

$$R = remainder\left[\frac{D \cdot 2^r}{G}\right]$$



# Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs
- addressing, ARP
- Ethernet
- switches
- VLANS

5.5 link virtualization: MPLS

5.6 data center networking

5.7 a day in the life of a web request

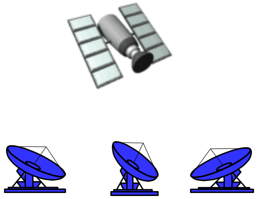
# Multiple access links, protocols

two types of "links":

- point-to-point
  - PPP for dial-up access
  - point-to-point link between Ethernet switch, host
- *broadcast (shared wire or medium)*
  - old-fashioned Ethernet
  - upstream HFC
  - 802.11 wireless LAN

shared wire (e.g., cabled Ethernet)

shared RF (e.g., 802.11 WiFi)



shared RF (satellite)

humans at a cocktail party (shared air, acoustical)

# Multiple access protocols

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
  - *collision* if node receives two or more signals at the same time

*multiple access protocol*

- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
  - no out-of-band channel for coordination

# An ideal multiple access protocol

*given:* broadcast channel of rate R bps

*desiderata:*

1. when one node wants to transmit, it can send at rate R.
2. when M nodes want to transmit, each can send at average rate R/M
3. fully decentralized:
   - no special node to coordinate transmissions
   - no synchronization of clocks, slots
4. simple

# MAC protocols: taxonomy

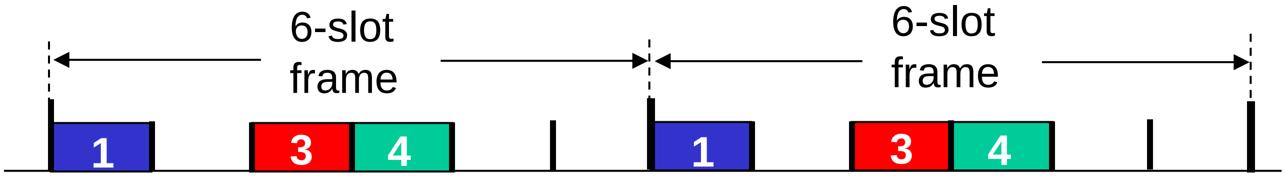
three broad classes:

- *channel partitioning*
  - divide channel into smaller "pieces" (time slots, frequency, code)
  - allocate piece to node for exclusive use
- *random access*
  - channel not divided, allow collisions
  - "recover" from collisions
- *"taking turns"*
  - nodes take turns, but nodes with more to send can take longer turns

# Channel partitioning MAC protocols: TDMA

## TDMA: time division multiple access

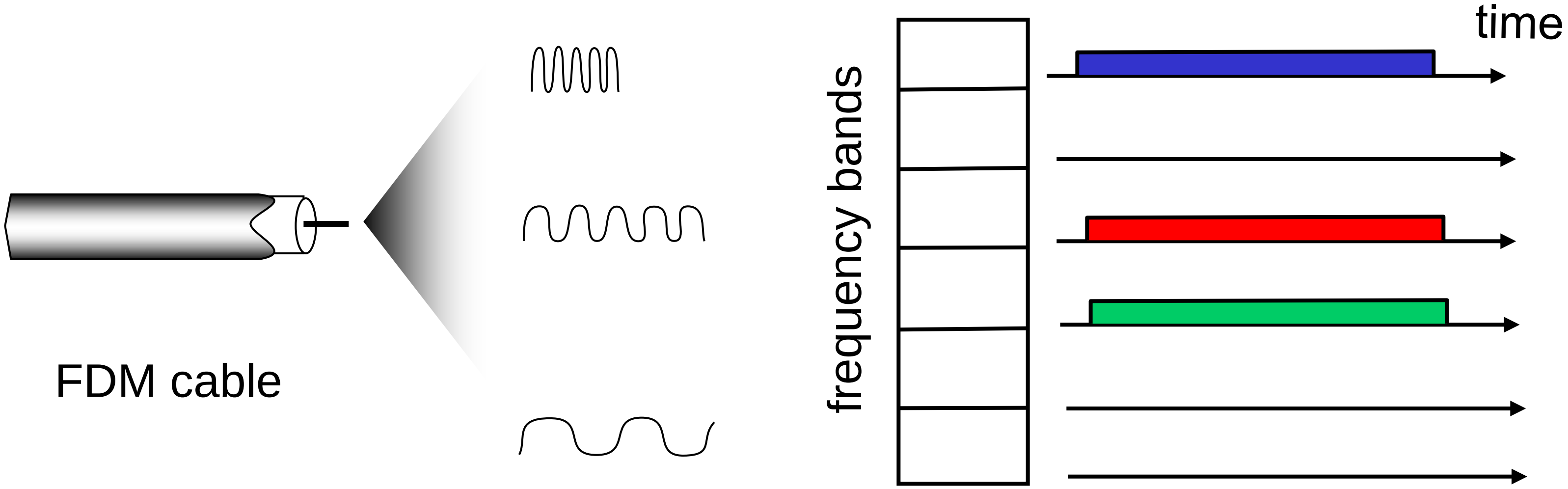
- access to channel in "rounds"
- each station gets fixed length slot (length = pkt trans time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have pkt, slots 2,5,6 idle



# Channel partitioning MAC protocols: FDMA

## FDMA: frequency division multiple access

- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have pkt, frequency bands 2,5,6 idle



# Random access protocols

- when node has packet to send
  - transmit at full channel data rate R.
  - no *a priori* coordination among nodes
- two or more transmitting nodes ➜ "collision"
- random access MAC protocol specifies:
  - how to detect collisions
  - how to recover from collisions (e.g., via delayed retransmissions)
- examples of random access MAC protocols:
  - slotted ALOHA
  - ALOHA
  - CSMA, CSMA/CD, CSMA/CA

# Slotted ALOHA

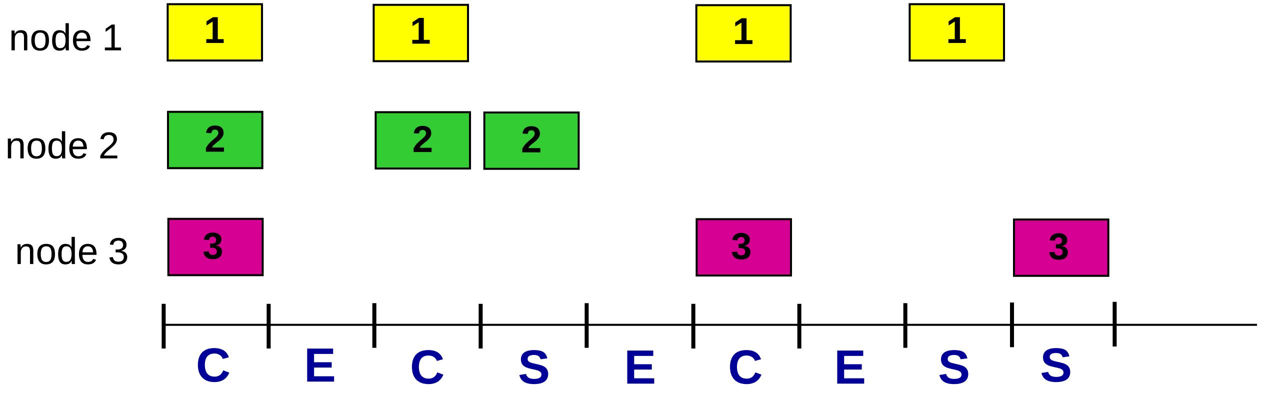
*assumptions:*

- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

*operation:*

- when node obtains fresh frame, transmits in next slot
  - *if no collision:* node can send new frame in next slot
  - *if collision:* node retransmits frame in each subsequent slot with prob. p until success

# Slotted ALOHA



*Pros:*

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

*Cons:*

- collisions, wasting slots
- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- clock synchronization

# Slotted ALOHA: efficiency

*efficiency*: long-run fraction of successful slots (many nodes, all with many frames to send)

- *suppose:* *N* nodes with many frames to send, each transmits in slot with probability *p*
- prob that given node has success in a slot = $p(1-p)^{N-1}$
- prob that *any* node has a success = $Np(1-p)^{N-1}$

- max efficiency: find p* that maximizes $Np(1-p)^{N-1}$
- for many nodes, take limit of $Np^*(1-p^*)^{N-1}$ as N goes to infinity, gives:

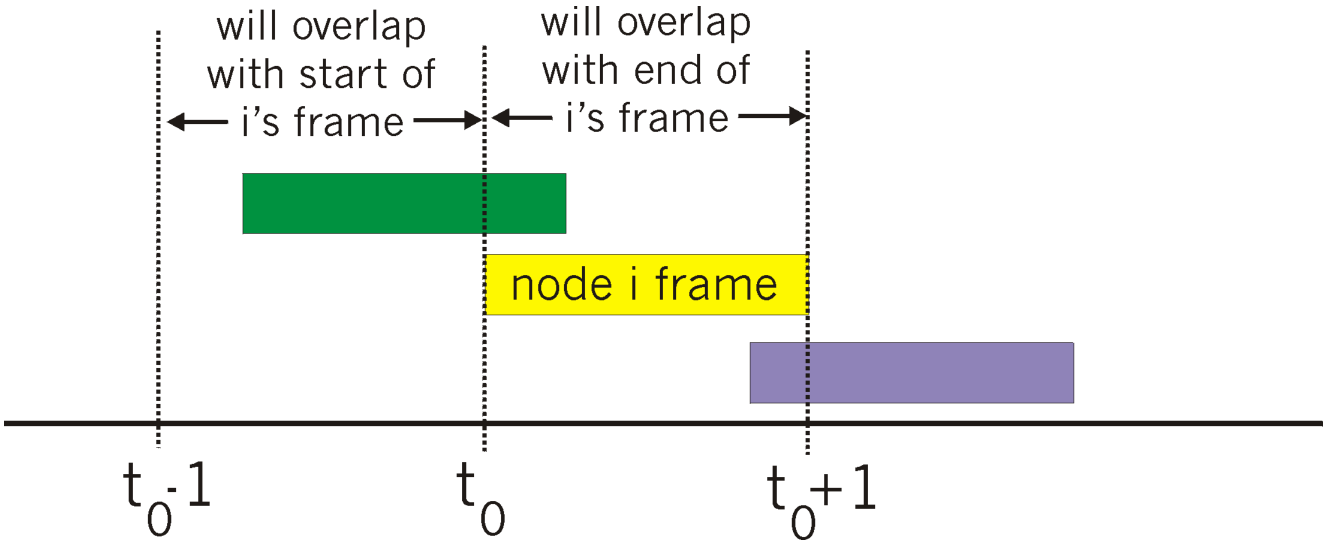
  *max efficiency = 1/e = 0.37*

*at best:* channel used for useful transmissions 37% of time!

!

# Pure (unslotted) ALOHA

- unslotted Aloha: simpler, no synchronization
- when frame first arrives
  - transmit immediately
- collision probability increases:
  - frame sent at $t_0$ collides with other frames sent in $[t_0-1, t_0+1]$


will overlap with start of i's frame
will overlap with end of i's frame
node i frame
$t_0$-1
$t_0$
$t_0$+1

# Pure ALOHA efficiency

P(success by given node) = P(node transmits) $\cdot$

P(no other node transmits in $[t_0\text{-}1, t_0]$ $\cdot$

P(no other node transmits in $[t_0, t_0+1]$

$$= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1}$$

$$= p \cdot (1-p)^{2(N-1)}$$

… choosing optimum p and then letting n → ∞

$$= 1/(2e) = 0.18$$

even *worse* than slotted Aloha!

# CSMA (carrier sense multiple access)

*CSMA*: listen before transmit:

if channel sensed idle: transmit entire frame

- if channel sensed busy, defer transmission

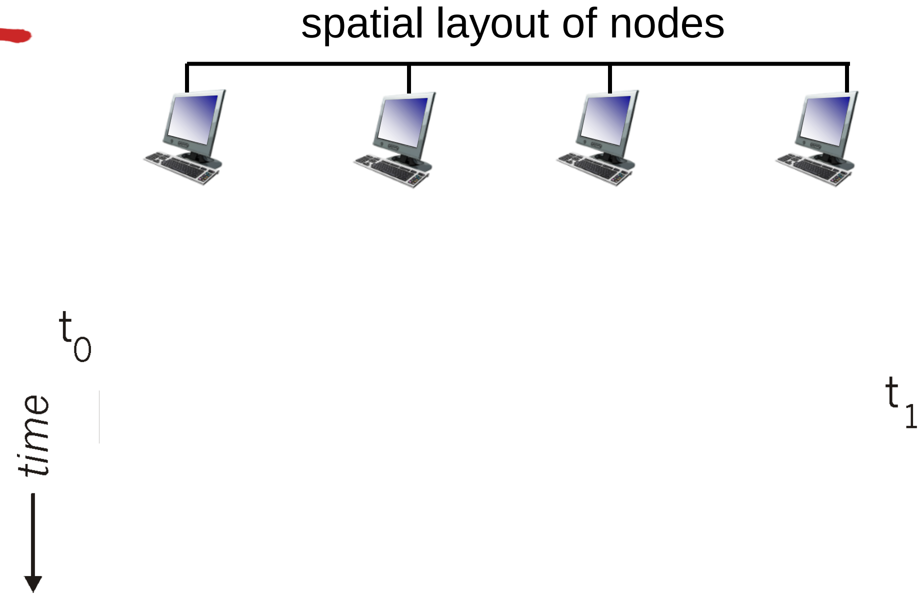
- human analogy: don’t interrupt others!

# CSMA collisions

- collisions *can* still occur: propagation delay means two nodes may not hear each other’s transmission
- collision: entire packet transmission time wasted
  - distance & propagation delay play role in in determining collision probability

# CSMA collisions

- collisions *can* still occur: propagation delay means two nodes may not hear each other’s transmission
- collision: entire packet transmission time wasted
  - distance & propagation delay play role in in determining collision probability



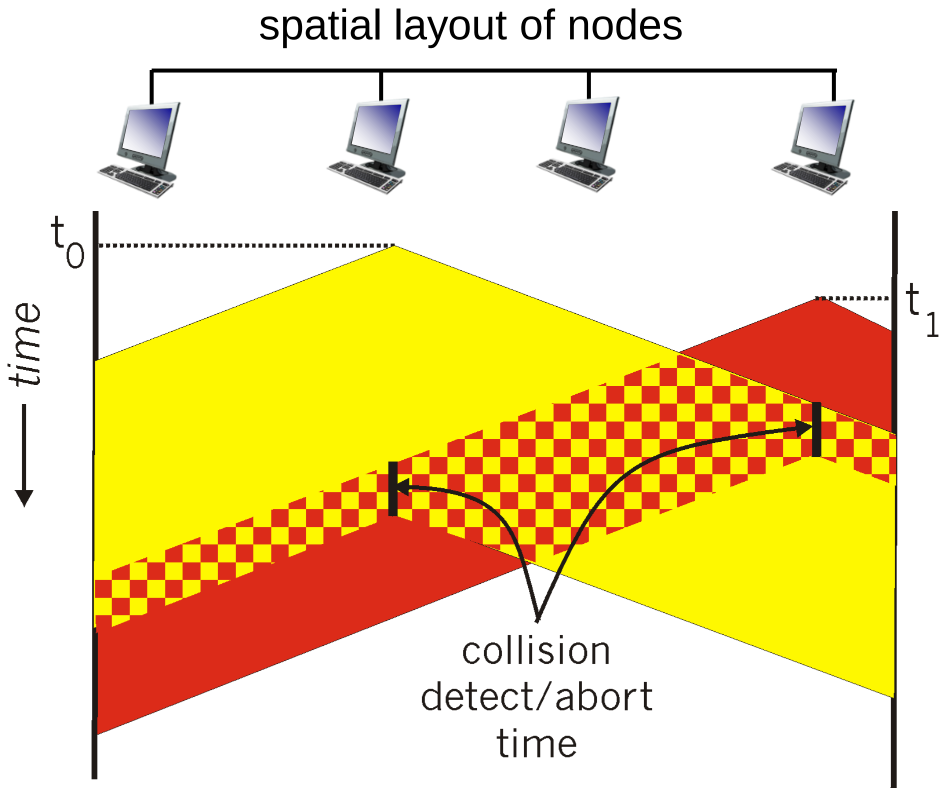
# CSMA/CD (collision detection)

*CSMA/CD:* carrier sensing, deferral as in CSMA

- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage

- collision detection:
  - easy in wired LANs: measure signal strengths, compare transmitted, received signals
  - difficult in wireless LANs: received signal strength overwhelmed by local transmission strength
- human analogy: the polite conversationalist

# CSMA/CD (collision detection)



# Ethernet CSMA/CD algorithm

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel idle, starts frame transmission. If NIC senses channel busy, waits until channel idle, then transmits.
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame !
4. If NIC detects another transmission while transmitting, aborts and sends jam signal
5. After aborting, NIC enters *binary (exponential) backoff:*
   - after $m$th collision, NIC chooses $K$ at random from $\{0,1,2, \ldots, 2^m-1\}$. NIC waits K·512 bit times, returns to Step 2
   - longer backoff interval with more collisions

# CSMA/CD efficiency

- $T_{prop}$ = max prop delay between 2 nodes in LAN
- $t_{trans}$ = time to transmit max-size frame

$$efficiency = \frac{1}{1 + 5t_{prop}/t_{trans}}$$

- efficiency goes to 1
  - as $t_{prop}$ goes to 0
  - as $t_{trans}$ goes to infinity
- better performance than ALOHA: and simple, cheap, decentralized!

# Example 5-1

在一个采用CSMA/CD协议的网络中，传输介质是一根完整的电缆，传输速率为1 Gbps，电缆中的信号传播速度是 200 000 km/s。若最小数据帧长度减少800比特，则最远的两个站点之间的距离至少需要

A.增加160 m　　B.增加80 m

C.减少160 m　　D.减少80 m

解：根据CSMA/CD协议工作原理，有

$L_{min}/R=2*D/V$,则$D=(V/2R)*L_{min}$，于是

$\Delta D=(V/2R)*\Delta L_{min}$

将V=200 000 km/s, R=1 Gbps, $\Delta L_{min}$ =-800bit,代入得：

$\Delta D=(200000*10^3/(2*10^9))*(-800)=-80$ m

答案：D

# "Taking turns" MAC protocols

## channel partitioning MAC protocols:

- share channel *efficiently* and *fairly* at high load
- inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

## random access MAC protocols

- efficient at low load: single node can fully utilize channel
- high load: collision overhead

## "taking turns" protocols

look for best of both worlds!

# “Taking turns” MAC protocols

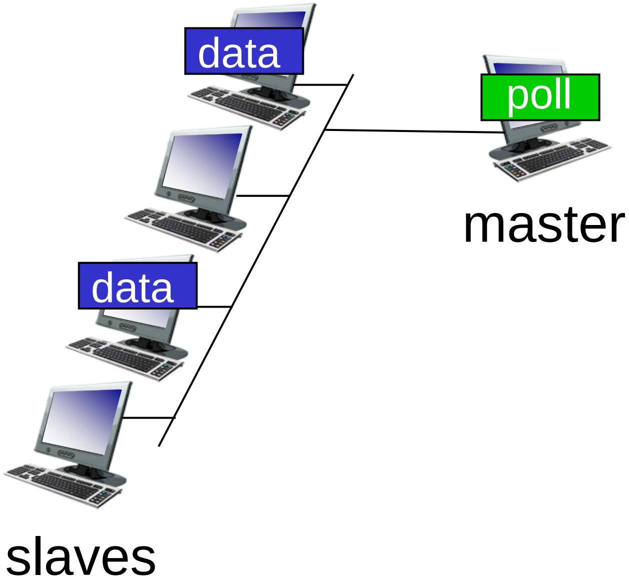
*polling:*

- master node “invites” slave nodes to transmit in turn
- typically used with “dumb” slave devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)

# "Taking turns" MAC protocols

*polling:*

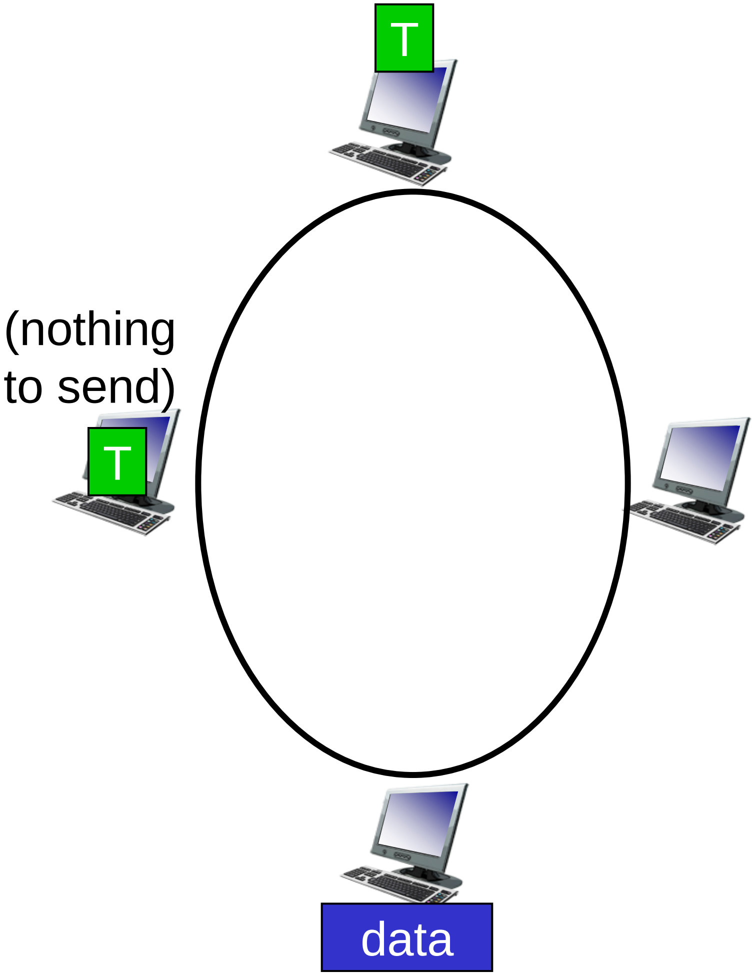
- master node "invites" slave nodes to transmit in turn
- typically used with "dumb" slave devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)



# "Taking turns" MAC protocols

*token passing:*

- control *token* passed from one node to next sequentially.
- token message
- concerns:
  - token overhead
  - latency
  - single point of failure (token)



# Summary of MAC protocols

- *channel partitioning,* by time, frequency or code
  - Time Division, Frequency Division
- *random access* (dynamic),
  - ALOHA, S-ALOHA, CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in Ethernet
  - CSMA/CA used in 802.11
- *taking turns*
  - polling from central site, token passing
  - bluetooth, FDDI, token ring

# Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

5.5 link virtualization: MPLS

5.6 data center networking

5.7 a day in the life of a web request

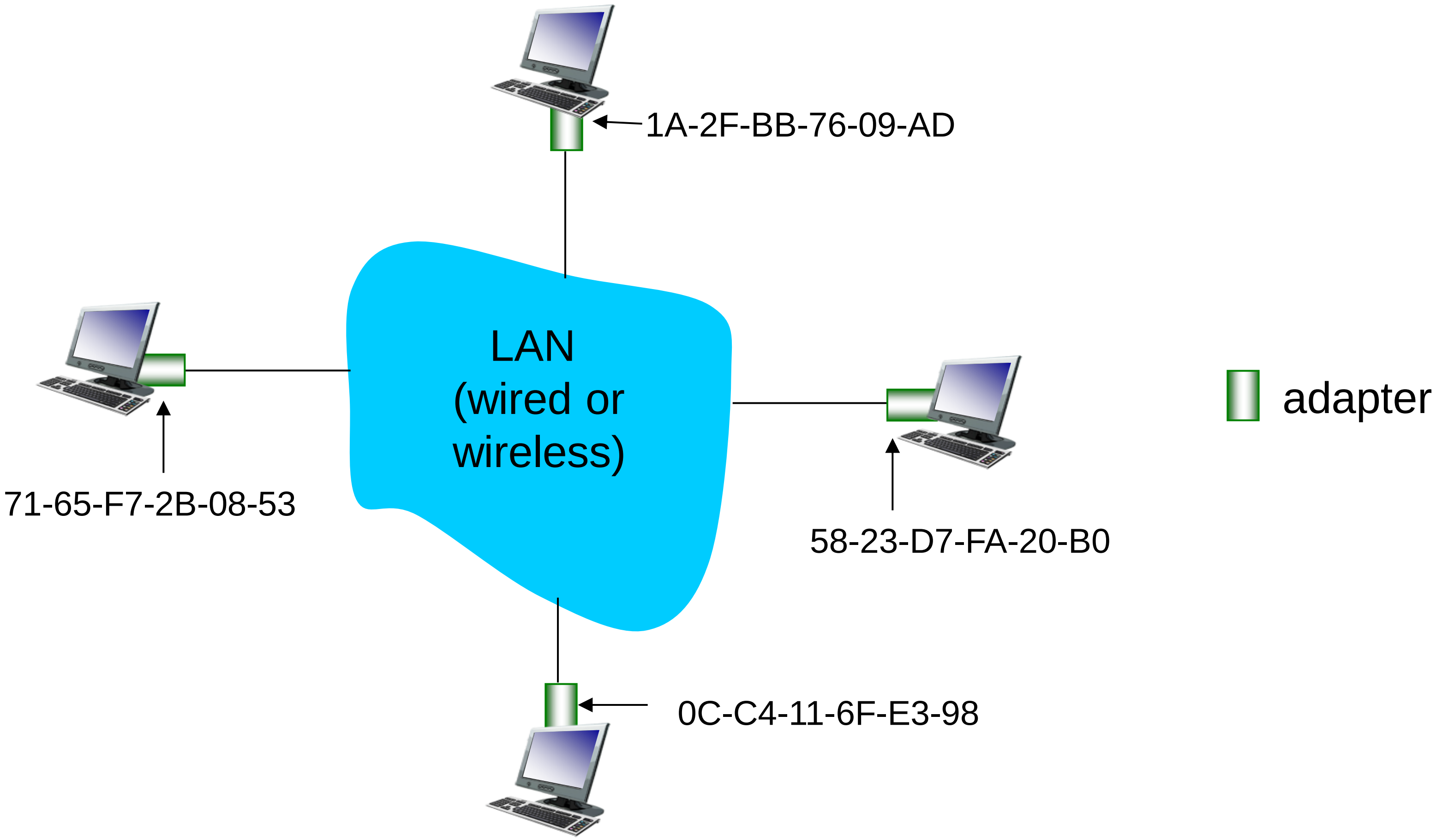
# MAC addresses and ARP

- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding
- MAC (or LAN or physical or Ethernet) address:
  - function: *used 'locally" to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)*
  - 48 bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
  - e.g.: 1A-2F-BB-76-09-AD

hexadecimal (base 16) notation
(each "number" represents 4 bits)

# LAN addresses and ARP

each adapter on LAN has unique *LAN* address

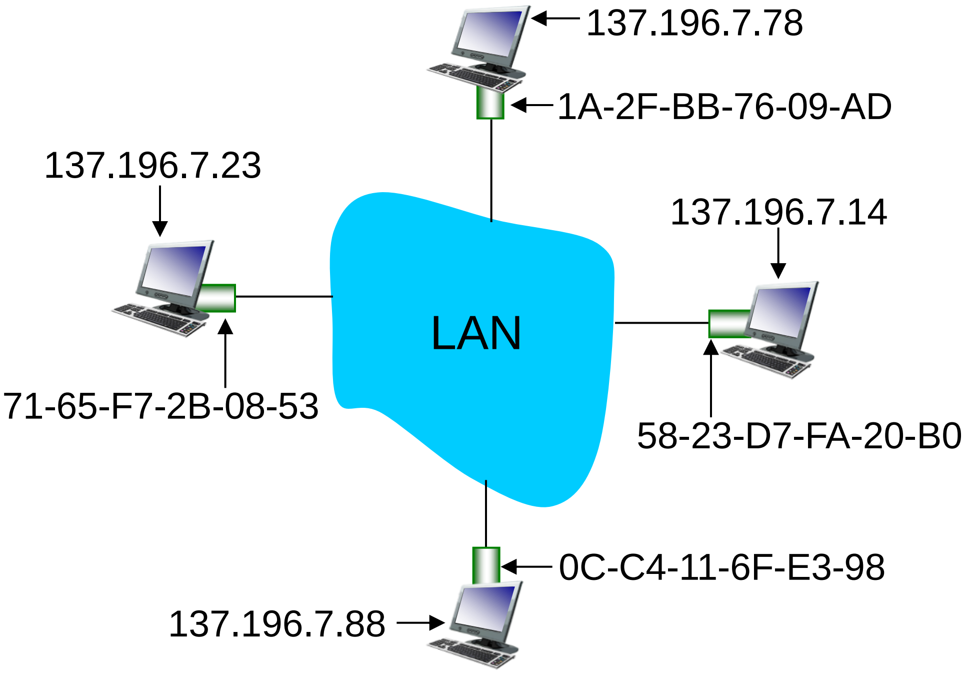


# LAN addresses (more)

- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- analogy:
  - MAC address: like Social Security Number
  - IP address: like postal address
- MAC flat address ➜ portability
  - can move LAN card from one LAN to another
- IP hierarchical address *not* portable
  - address depends on IP subnet to which node is attached

# ARP: address resolution protocol

*Question:* how to determine interface's MAC address, knowing its IP address?



*ARP table:* each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes:

  < IP address; MAC address; TTL>

- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

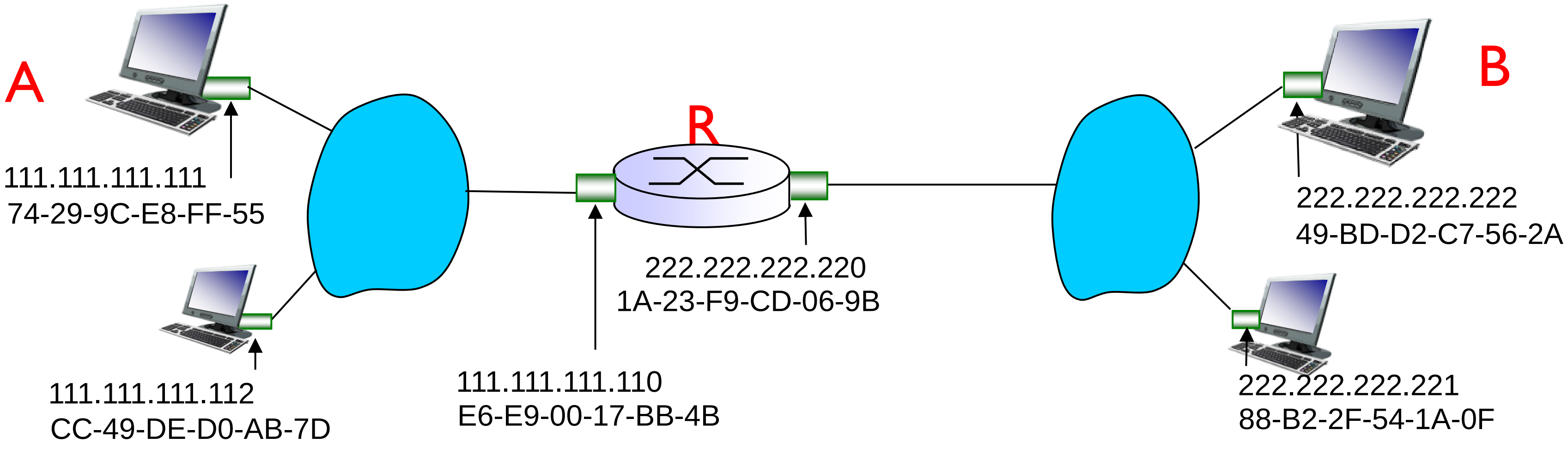
# ARP protocol: same LAN

- A wants to send datagram to B
  - B’s MAC address not in A’s ARP table.
- A broadcasts ARP query packet, containing B's IP address
  - dest MAC address = FF-FF-FF-FF-FF-FF
  - all nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
  - frame sent to A’s MAC address (unicast)
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
  - soft state: information that times out (goes away) unless refreshed
- ARP is “plug-and-play”:
  - nodes create their ARP tables *without intervention from net administrator*

# Addressing: routing to another LAN

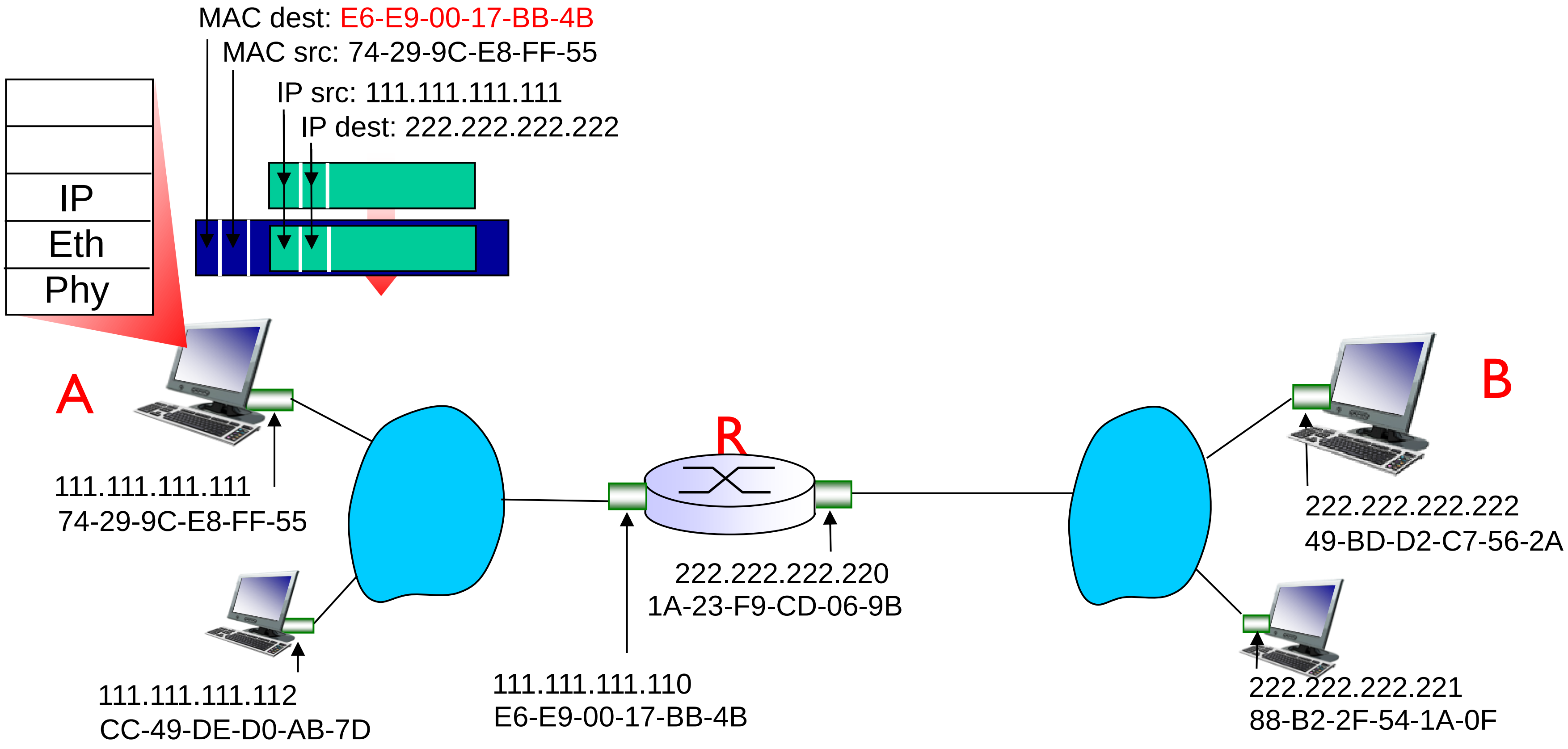
walkthrough: send datagram from A to B via R

- focus on addressing – at IP (datagram) and MAC layer (frame)
- assume A knows B’s IP address
- assume A knows IP address of first hop router, R (how?)
- assume A knows R’s MAC address (how?)



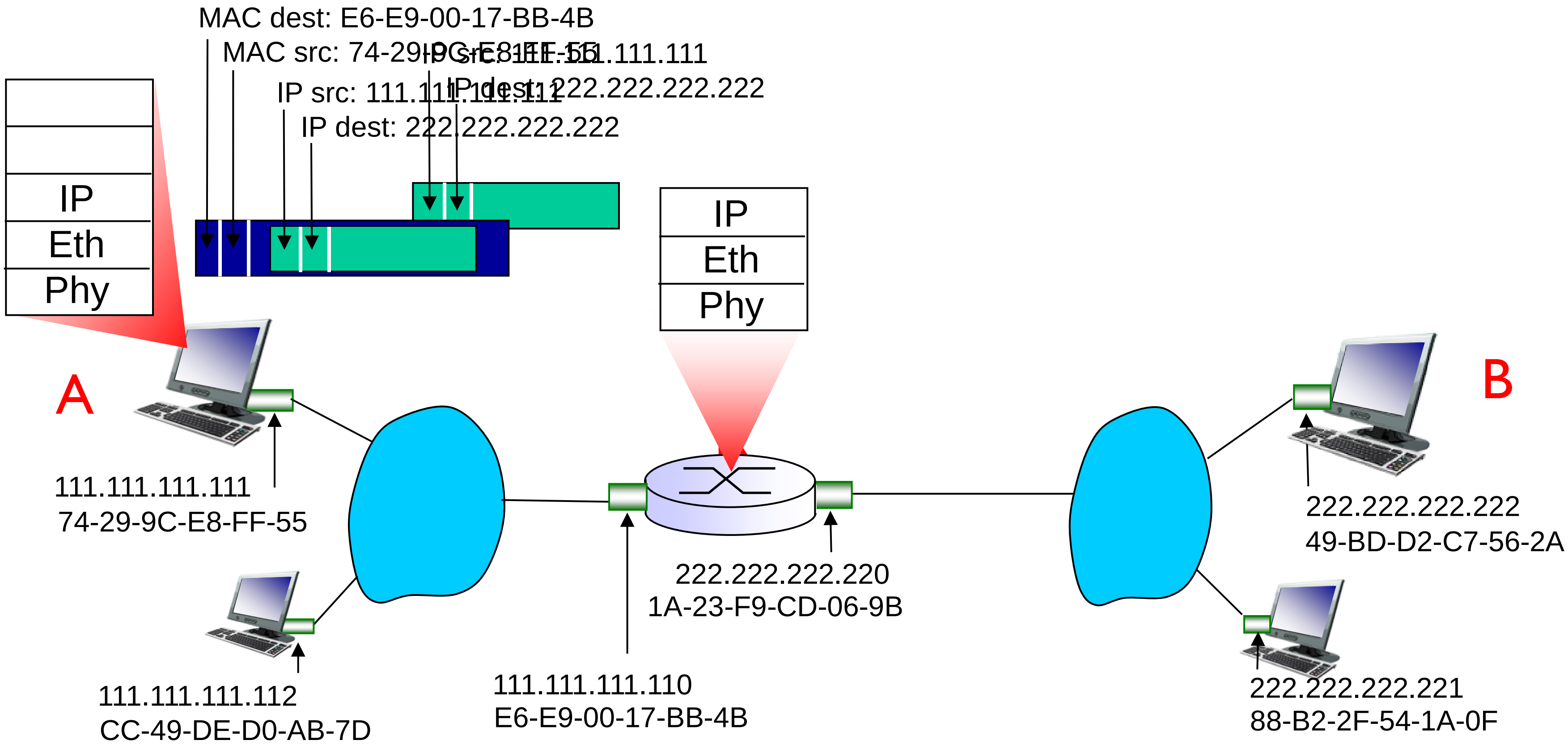
# Addressing: routing to another LAN

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as dest, frame contains A-to-B IP datagram



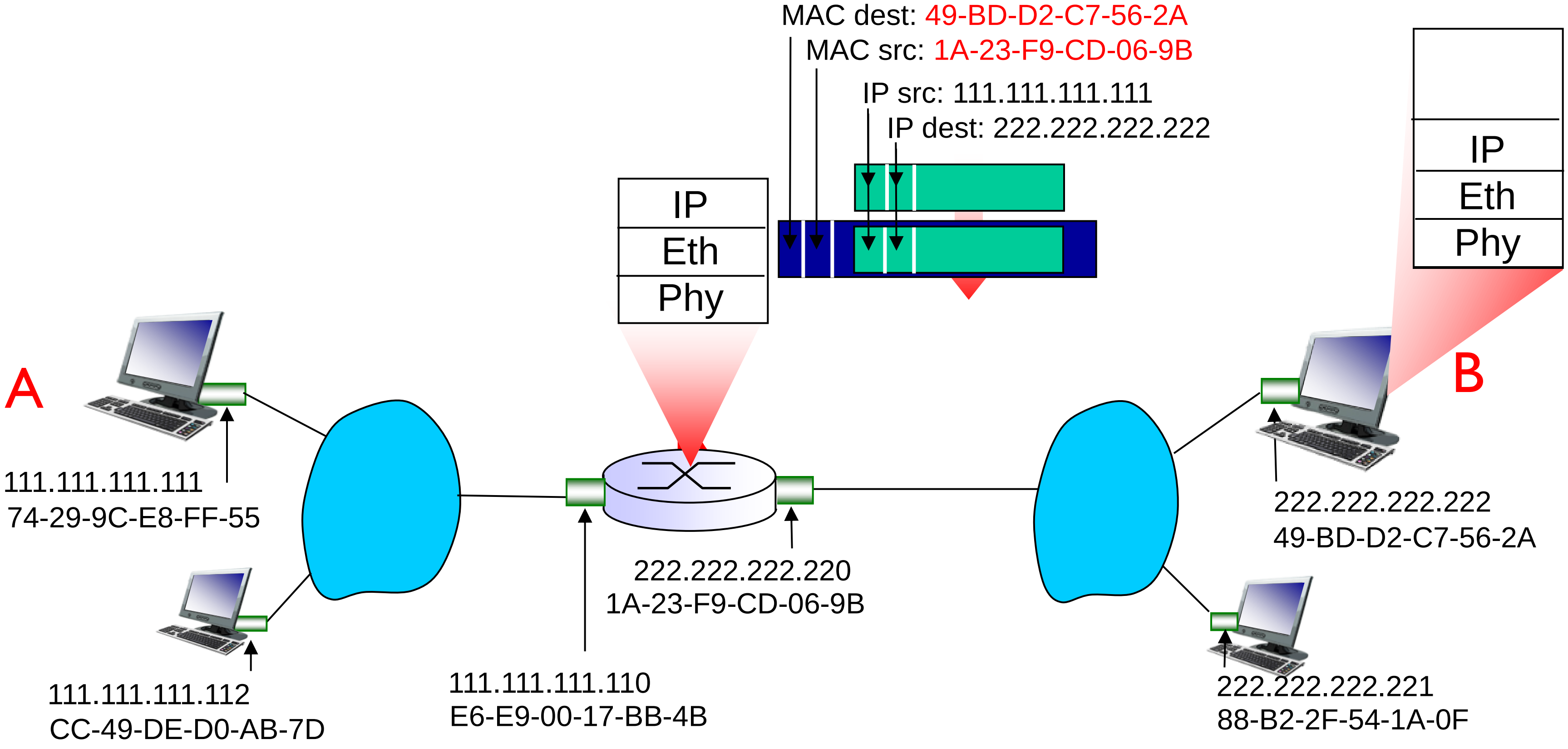
# Addressing: routing to another LAN

- frame sent from A to R
- frame received at R, datagram removed, passed up to IP



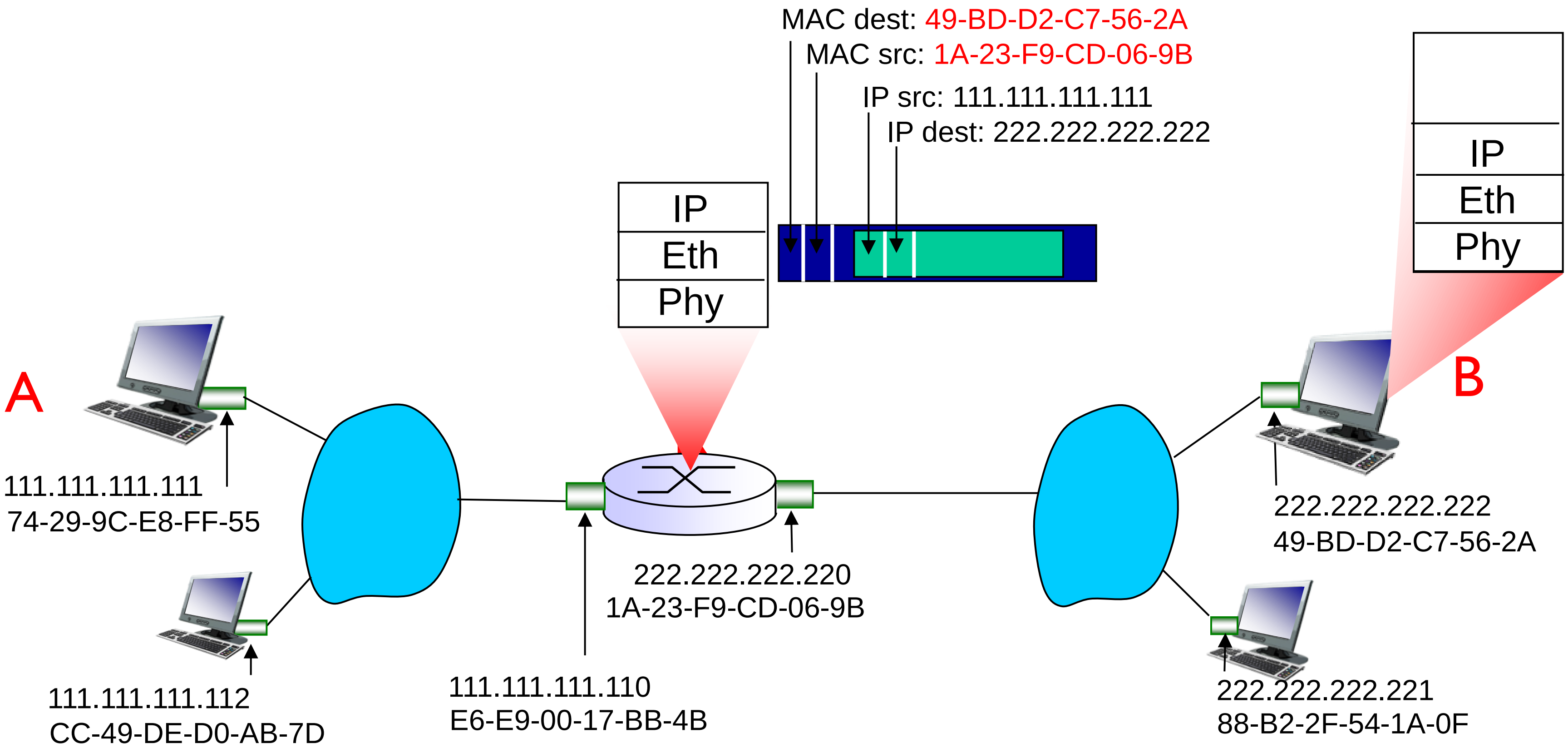
# Addressing: routing to another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



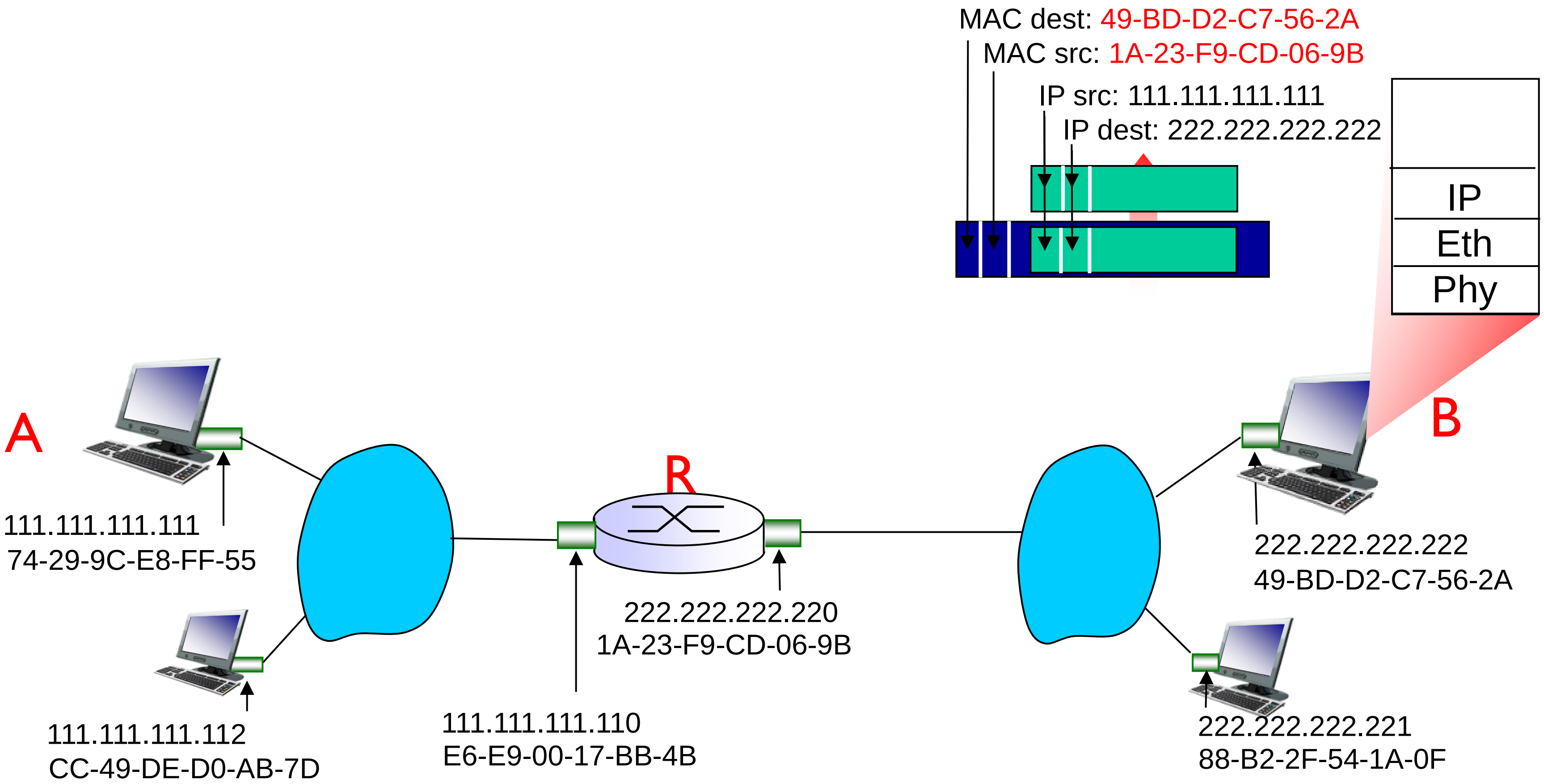
# Addressing: routing to another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



# Addressing: routing to another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



# Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

5.5 link virtualization: MPLS

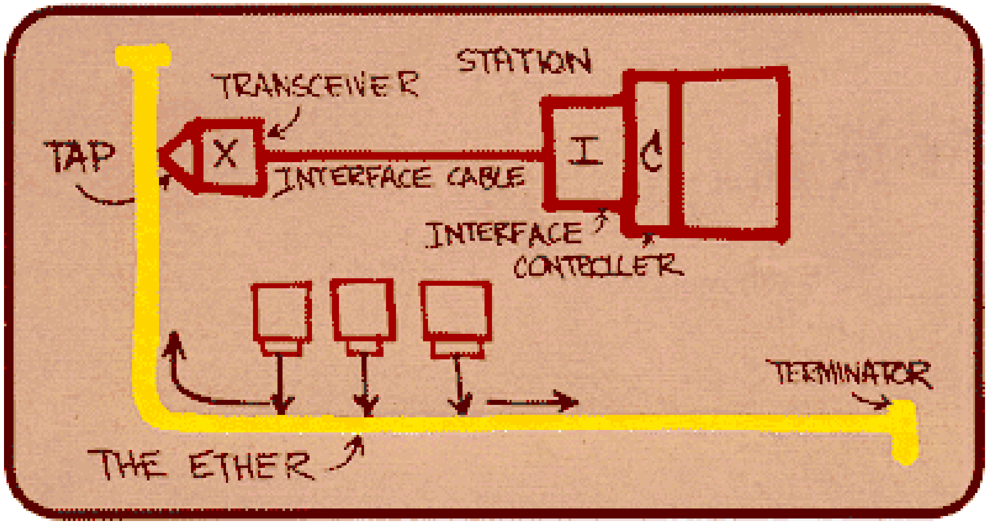
5.6 data center networking

5.7 a day in the life of a web request

# Ethernet

"dominant" wired LAN technology:

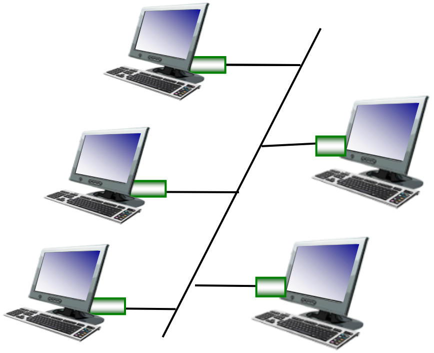
- cheap $20 for NIC
- first widely used LAN technology
- simpler, cheaper than token LANs and ATM
- kept up with speed race: 10 Mbps – 10 Gbps



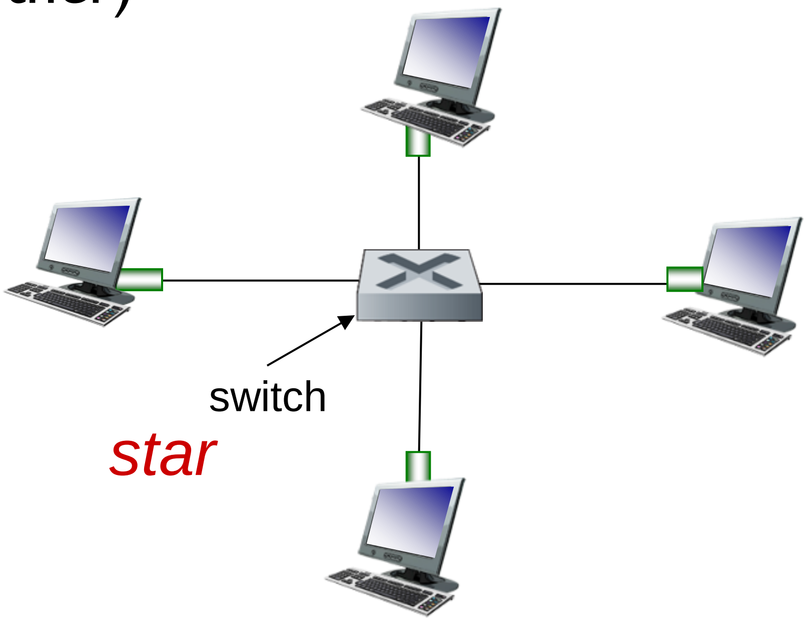
*Metcalfe's Ethernet sketch*

# Ethernet: physical topology

- *bus:* popular through mid 90s
  - all nodes in same collision domain (can collide with each other)
- *star:* prevails today
  - active *switch* in center
  - each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)



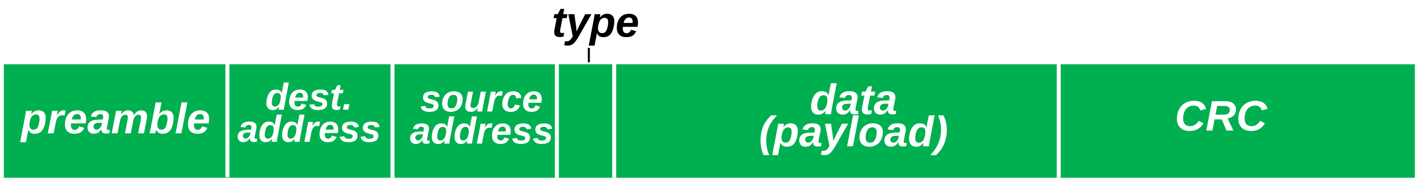
*bus:* coaxial cable



*star*

# Ethernet frame structure

sending adapter encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame

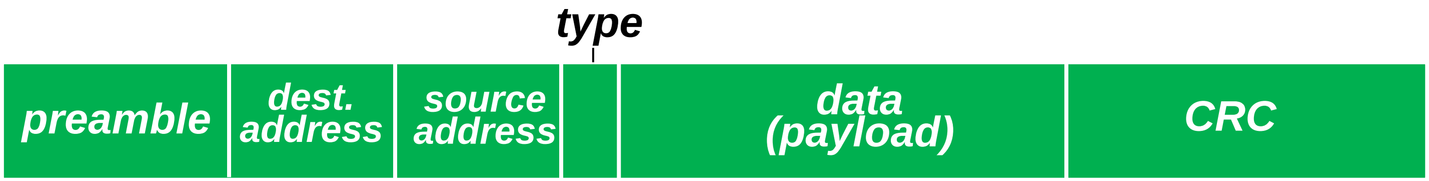


*Preamble(8B):*

- 7 bytes with pattern 10101010 followed by one byte with pattern 10101011
- used to synchronize receiver, sender clock rates

# Ethernet frame structure (more)

- *addresses:* 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address (e.g. ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- *Type(2B):* indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX, AppleTalk)
- *CRC(4B):* cyclic redundancy check at receiver
  - error detected: frame is dropped



type

preamble

dest. address

source address

data (payload)

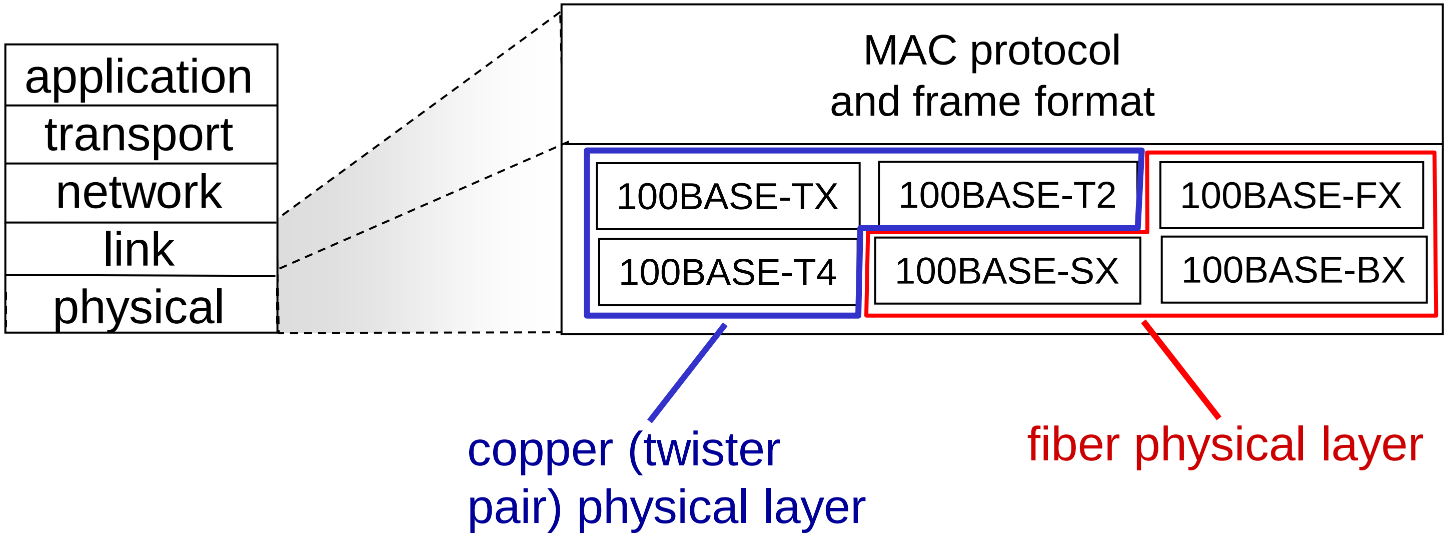
CRC

# Ethernet: unreliable, connectionless

- *connectionless:* no handshaking between sending and receiving NICs
- *unreliable:* receiving NIC doesnt send acks or nacks to sending NIC
  - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- Ethernet's MAC protocol: unslotted *CSMA/CD wth binary backoff*

# 802.3 Ethernet standards: link & physical layers

- *many* different Ethernet standards
  - common MAC protocol and frame format
  - different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10G bps
  - different physical layer media: fiber, cable



# Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

5.5 link virtualization: MPLS

5.6 data center networking

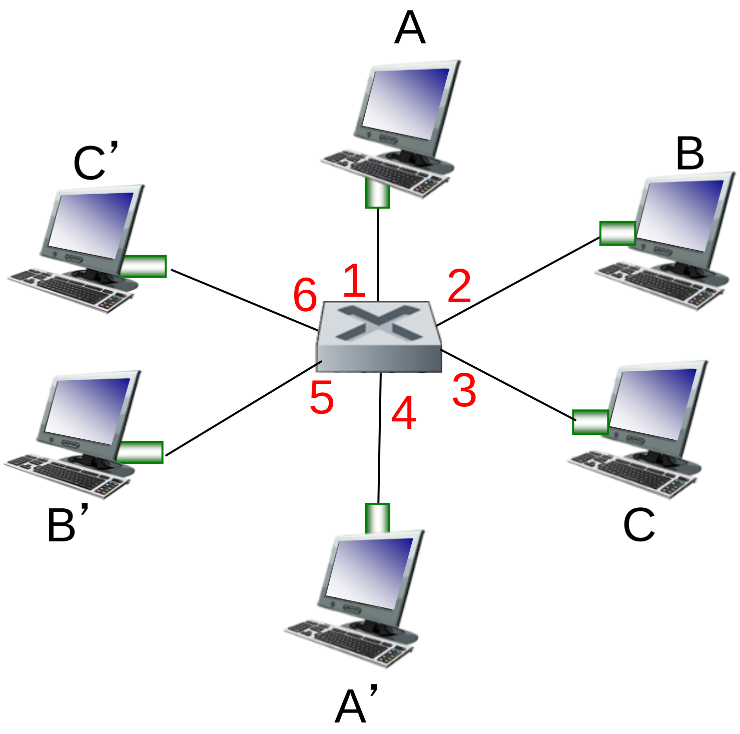
5.7 a day in the life of a web request

# Ethernet switch

- link-layer device: takes an *active* role
  - store, forward Ethernet frames
  - examine incoming frame’s MAC address, selectively forward  frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- *transparent*
  - hosts are unaware of presence of switches
- *plug-and-play, self-learning*
  - switches do not need to be configured

# Switch: *multiple* simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, but no collisions; full duplex
  - each link is its own collision domain
- *switching:* A-to-A' and B-to-B' can transmit simultaneously, without collisions

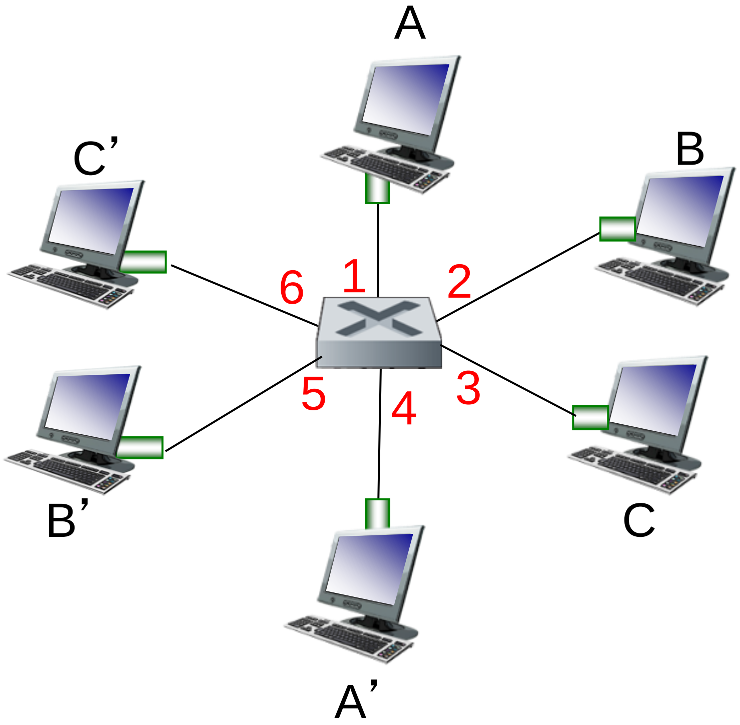


*switch with six interfaces (1,2,3,4,5,6)*

# Switch forwarding table

Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

- *A: each switch has a switch table, each entry:*
  - *(MAC address of host, interface to reach host, time stamp)*
  - *looks like a routing table!*



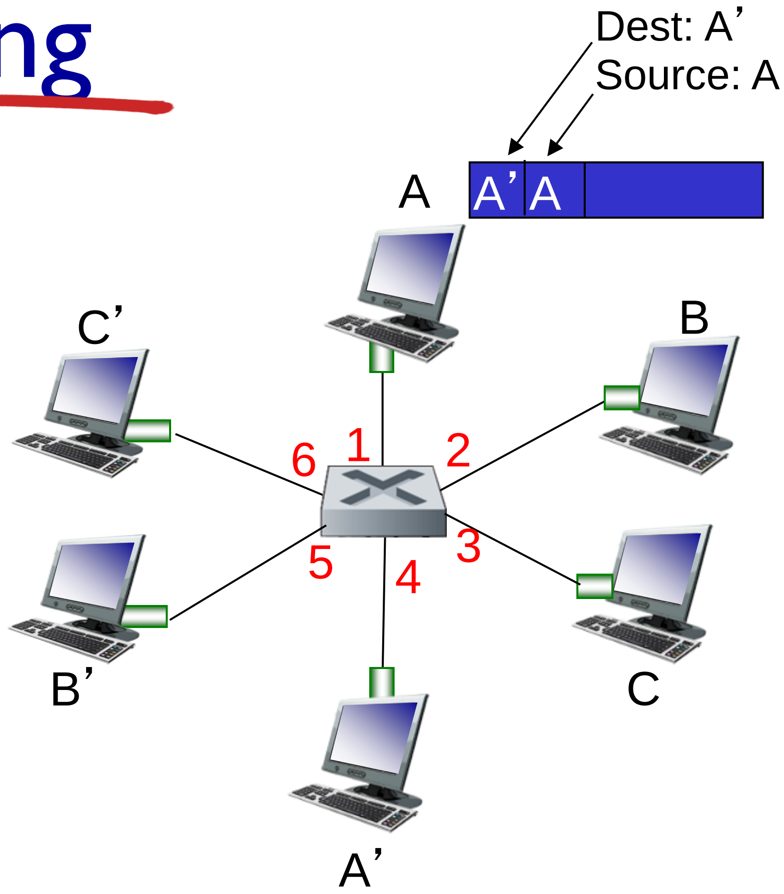
*switch with six interfaces (1,2,3,4,5,6)*

*Q: how are entries created, maintained in switch table?*

- *something like a routing protocol?*

# Switch: self-learning

- switch *learns* which hosts can be reached through which interfaces
  - when frame received, switch "learns" location of sender: incoming LAN segment
  - records sender/location pair in switch table



| MAC addr | interface | TTL |
|---|---|---|
| *A* | *1* | *60* |

*Switch table (initially empty)*

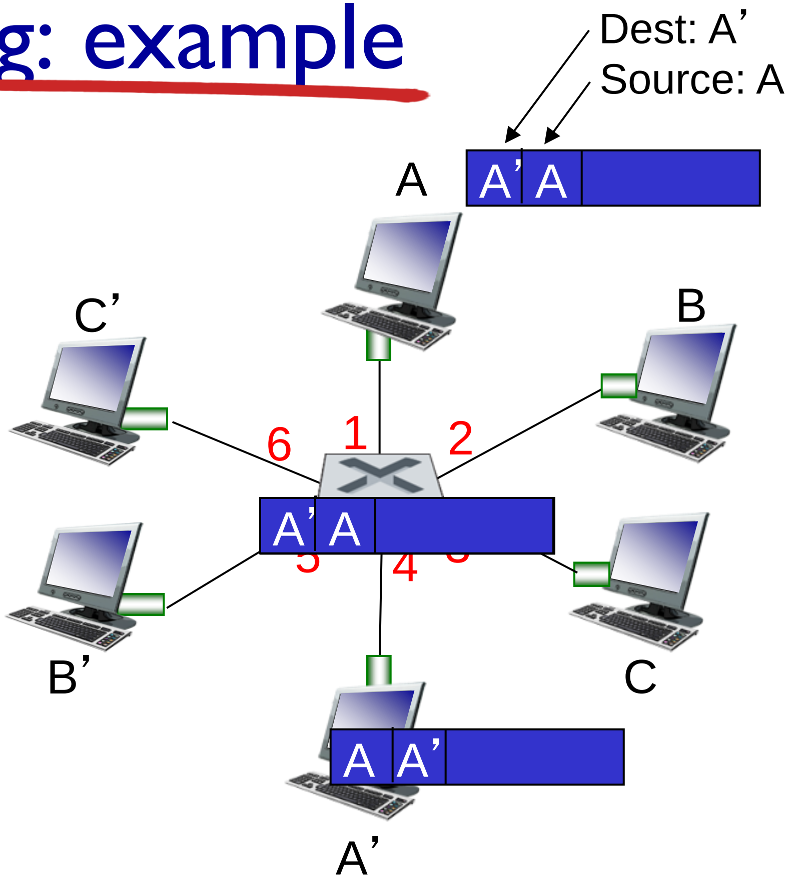
# Switch: frame filtering/forwarding

when frame received at switch:

```
1. record incoming link, MAC address of sending host
2. index switch table using MAC destination address
3. if entry found for destination
      then {
      if destination on segment from which frame arrived
          then drop frame
          else forward frame on interface indicated by entry
       }
      else flood  /* forward on all interfaces except arriving
                      interface */
```

# Self-learning, forwarding: example

- frame destination, A', locaton unknown: *flood*
- destination A location known: *selectively send on just one link*

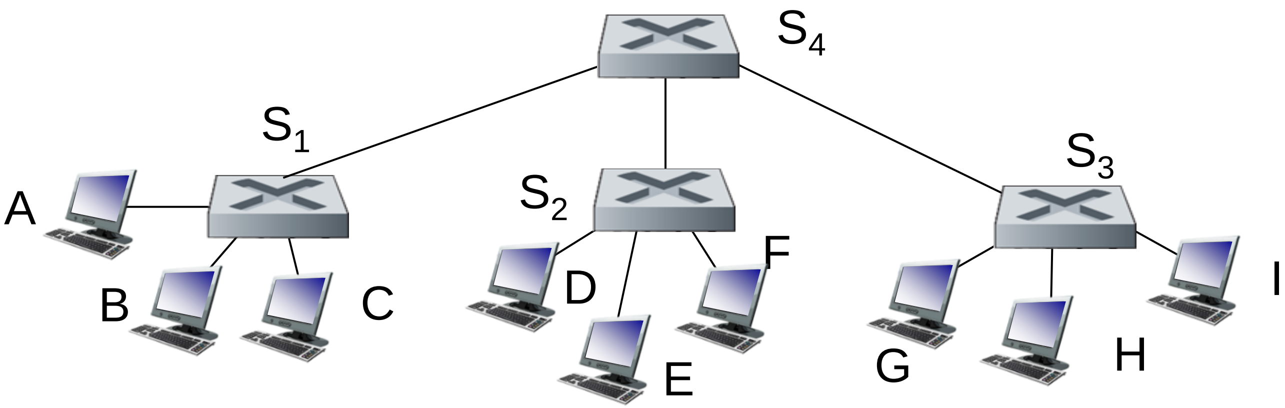


| MAC addr | interface | TTL |
|---|---|---|
| *A* | *1* | *60* |
| *A'* | *4* | *60* |

*switch table (initially empty)*

# Interconnecting switches

- switches can be connected together

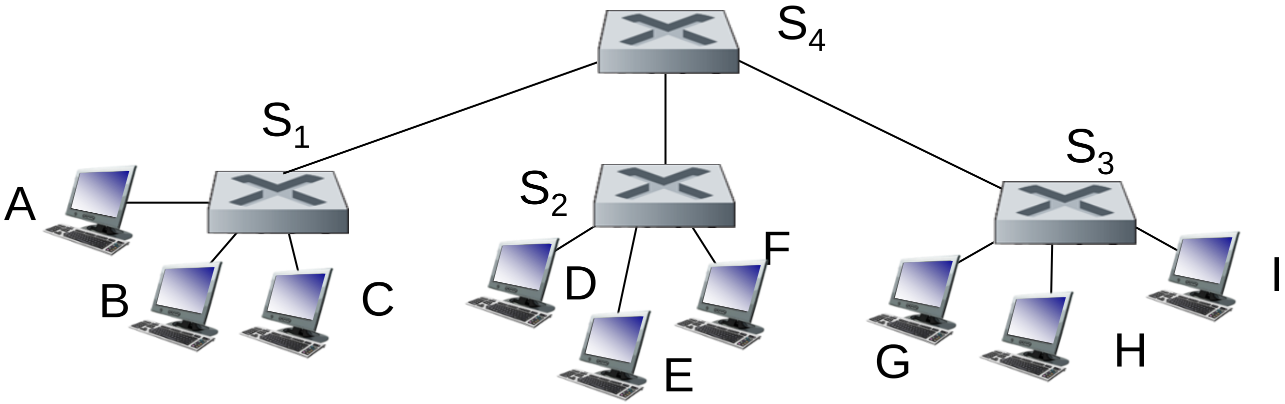


*Q:* sending from A to G - how does S$_1$ know to forward frame destined to F via S$_4$ and S$_3$?

- *A:* self learning! (works *exactly* the same as in single-switch case!)

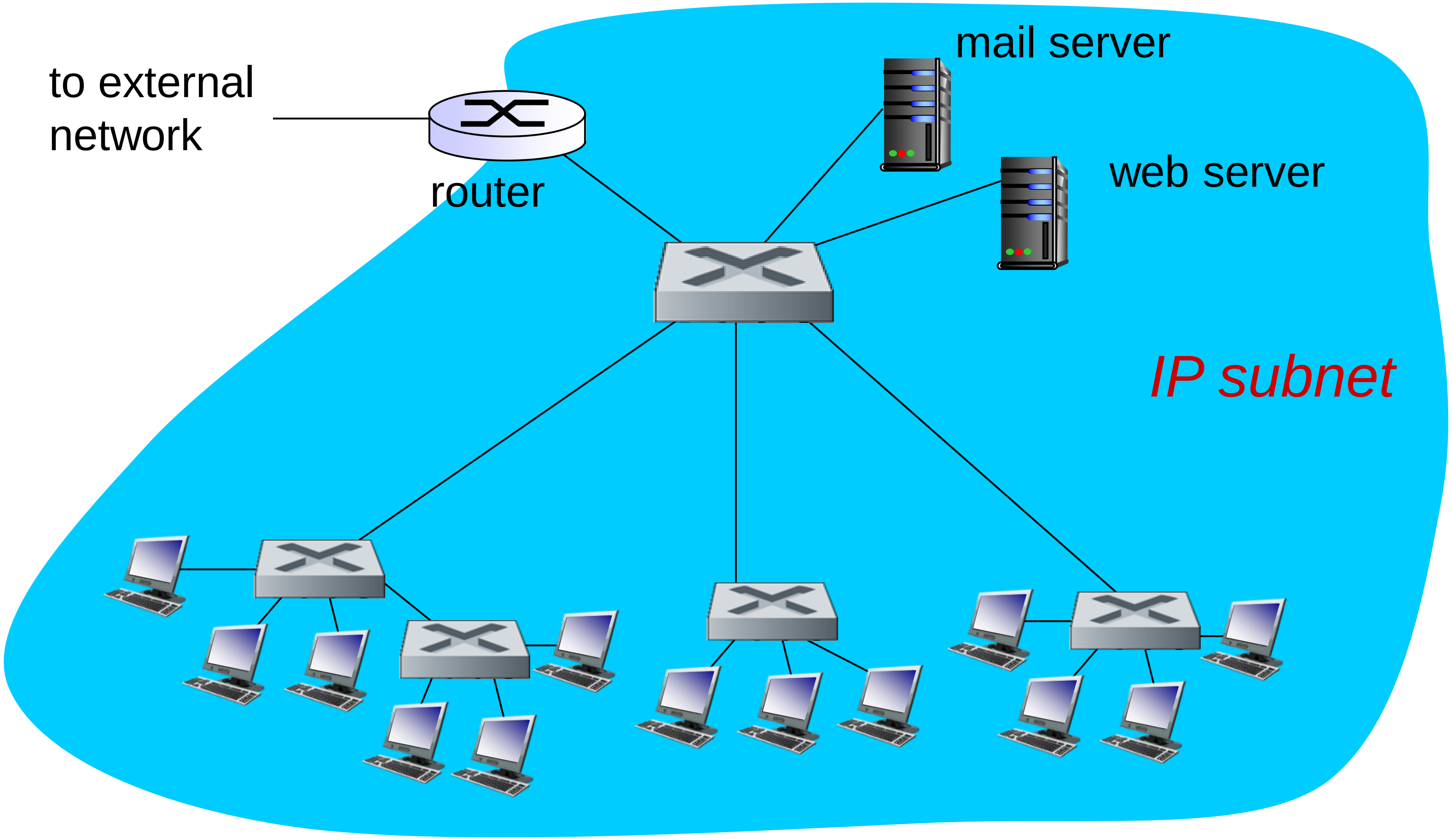
# Self-learning multi-switch example

Suppose C sends frame to I, I responds to C



- *Q:* show switch tables and packet forwarding in $S_1$, $S_2$, $S_3$, $S_4$

# Institutional network



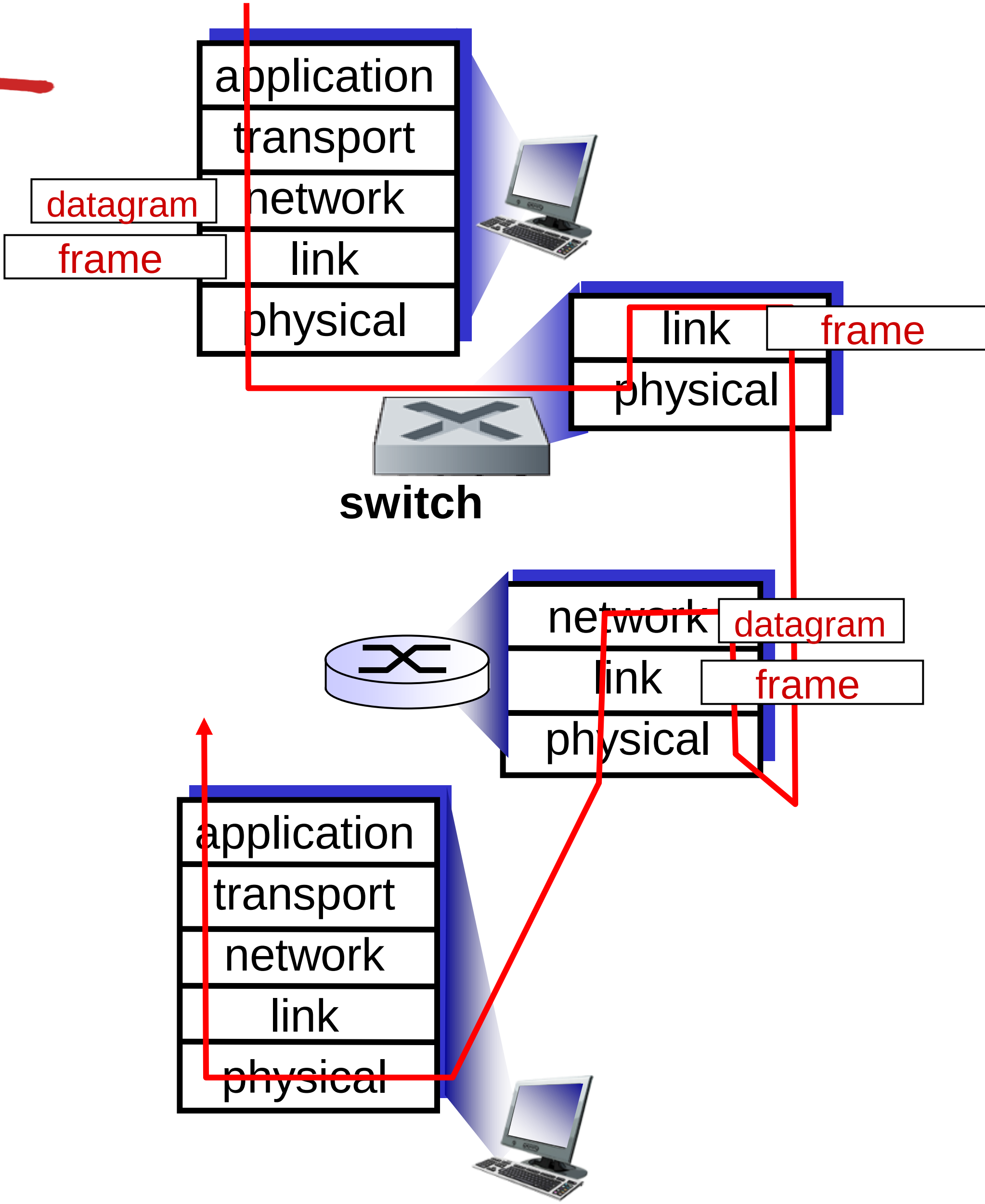
# Switches vs. routers

both are store-and-forward:

- *routers:* network-layer devices (examine network-layer headers)
- *switches*: link-layer devices (examine link-layer headers)

both have forwarding tables:

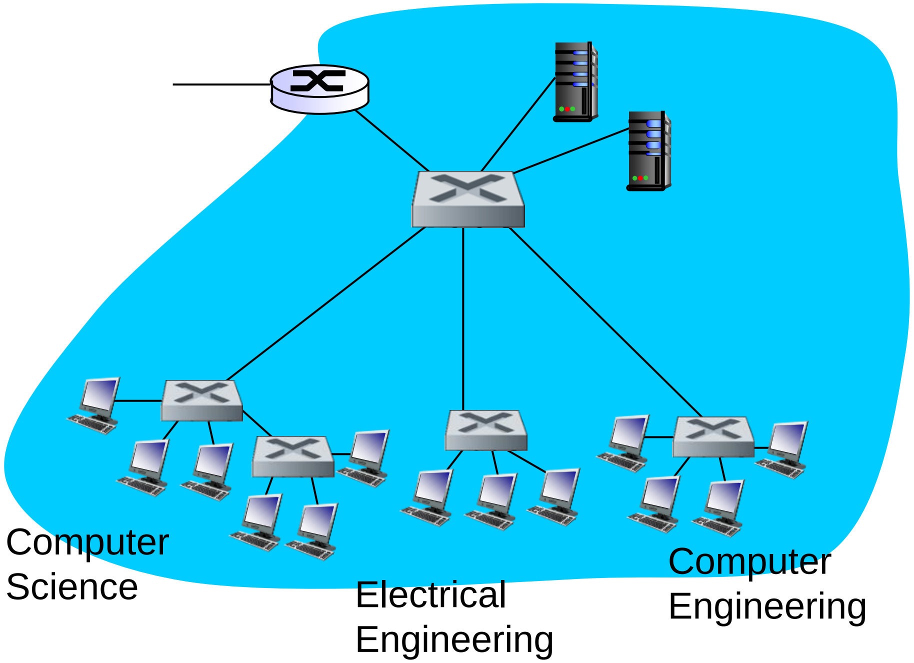
- *routers:* compute tables using routing algorithms, IP addresses
- *switches:* learn forwarding table using flooding, learning, MAC addresses



# Summary comparison

| | hubs | switches | bridges | routers |
|---|---|---|---|---|
| Layer | 1 | 2 | 2 | 3 |
| Traffic isolation | no | yes | yes | yes |
| Storm isolation | no | no | no | yes |
| Plug & play | yes | yes | yes | no |
| Optimal routing | no | no | no | yes |
| Cut through | yes | yes | yes | no |

# VLANs: motivation



*consider:*

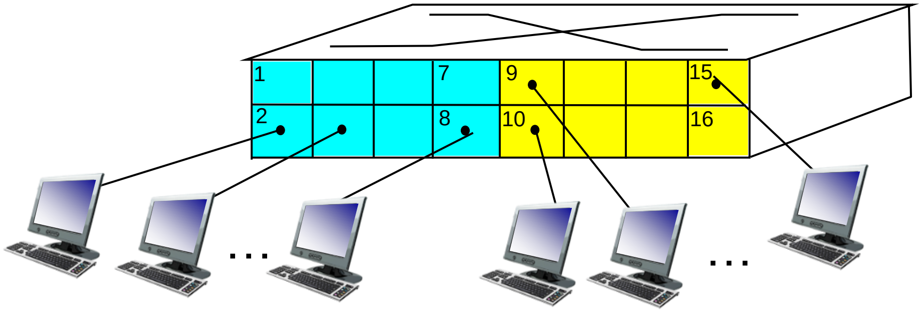
- CS user moves office to EE, but wants connect to CS switch?
- single broadcast domain:
  - all layer-2 broadcast traffic (ARP, DHCP, unknown location of destination MAC address) must cross entire LAN
  - security/privacy, efficiency issues

# VLANs

***Virtual Local Area Network***

switch(es) supporting VLAN capabilities can be configured to define multiple ***virtual*** LANS over single physical LAN infrastructure.

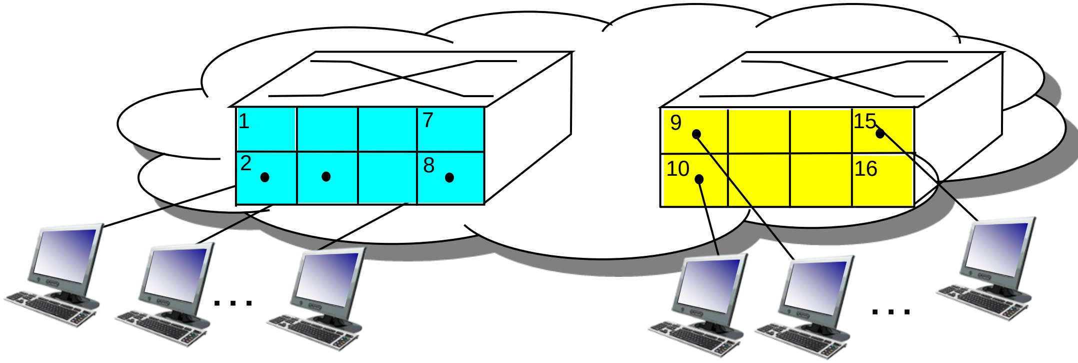
port-based VLAN: switch ports grouped (by switch management software) so that *single* physical switch ……



Electrical Engineering
(VLAN ports 1-8)

Computer Science
(VLAN ports 9-15)

… operates as *multiple* virtual switches

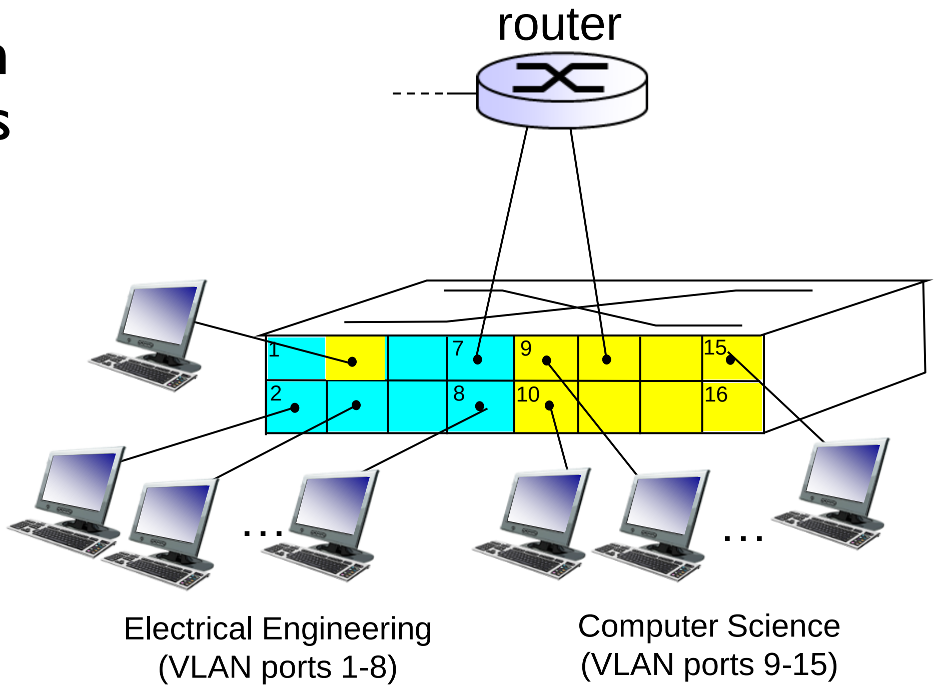


Electrical Engineering
(VLAN ports 1-8)

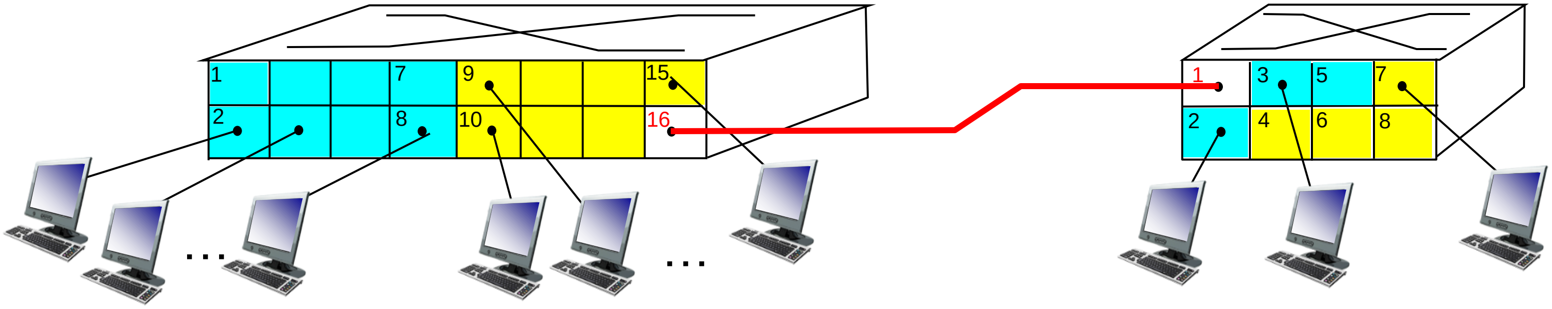
Computer Science
(VLAN ports 9-16)

# Port-based VLAN

- *traffic isolation:* frames to/from ports 1-8 can *only* reach ports 1-8
  - can also define VLAN based on MAC addresses of endpoints, rather than switch port
- *dynamic membership:* ports can be dynamically assigned among VLANs
- *forwarding between VLANS:* done via routing (just as with separate switches)
  - in practice vendors sell combined switches plus routers



# VLANS spanning multiple switches



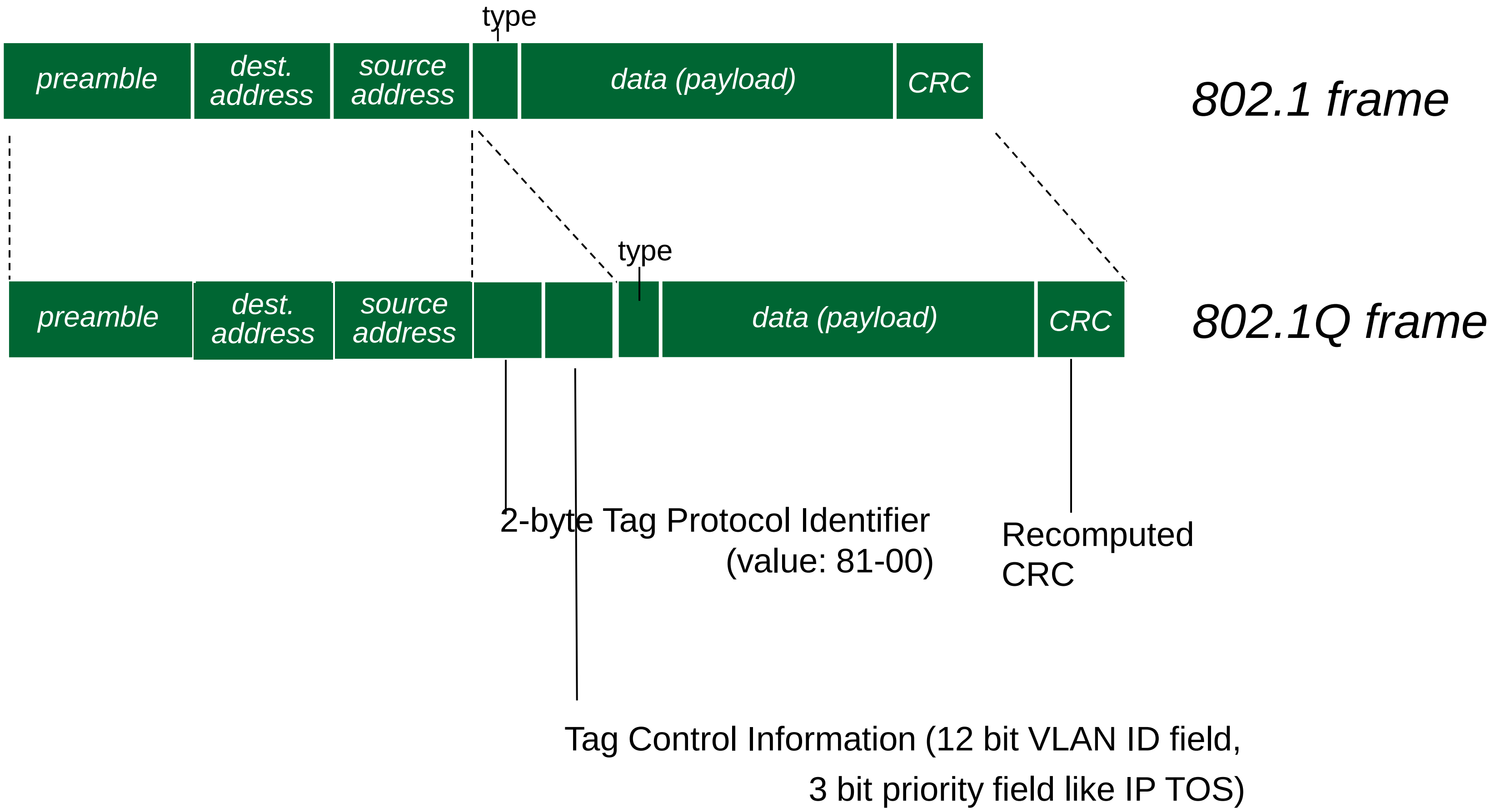
Electrical Engineering
(VLAN ports 1-8)

Computer Science
(VLAN ports 9-15)

Ports 2,3,5 belong to EE VLAN
Ports 4,6,7,8 belong to CS VLAN

- *trunk port:* carries frames between VLANS defined over multiple physical switches
  - frames forwarded within VLAN between switches can’t be vanilla 802.1 frames (must carry VLAN ID info)
  - 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports

# 802.1Q VLAN frame format



2-byte Tag Protocol Identifier (value: 81-00)

Recomputed CRC

Tag Control Information (12 bit VLAN ID field, 3 bit priority field like IP TOS)

# Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

5.5 PPP

5.6 link virtualization: MPLS

5.7 a day in the life of a web request

# Point to Point Data Link Control

- one sender, one receiver, one link: easier than broadcast link:
  - no Media Access Control
  - no need for explicit MAC addressing
  - e.g., dialup link, ISDN line
- popular  point-to-point DLC protocols:
  - HDLC: High level data link control (Data link used to be considered “high layer” in protocol stack!
  - PPP (point-to-point protocol)

# PPP Design Requirements [RFC 1557]

- packet framing: encapsulation of network-layer datagram in data link frame
  - carry network layer data of any network layer protocol (not just IP) *at same time*
  - ability to demultiplex upwards
- bit transparency: must carry any bit pattern in the data field
- error detection (no correction)
- connection liveness: detect, signal link failure to network layer
- network layer address negotiation: endpoint can learn/configure each other's network address

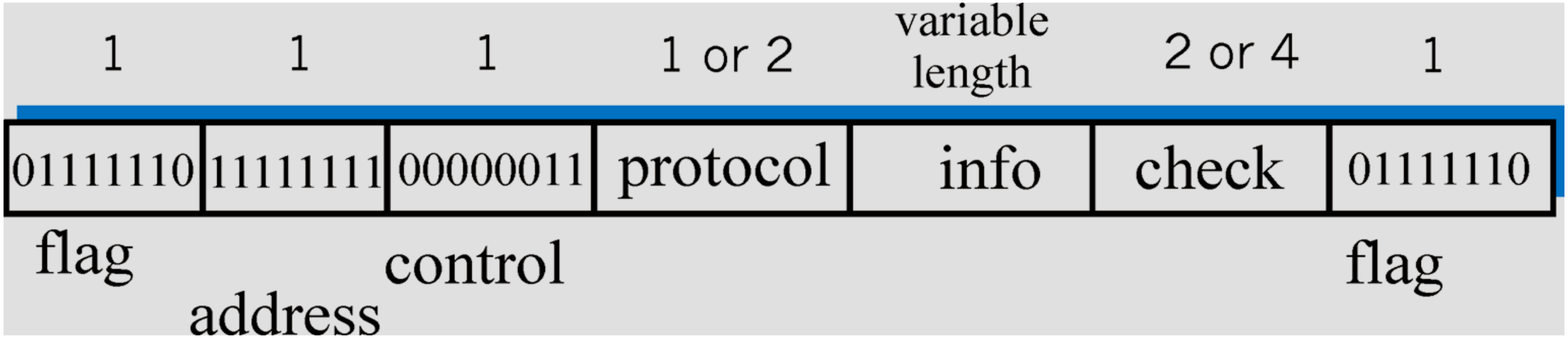
# PPP non-requirements

- no error correction/recovery
- no flow control
- out of order delivery OK
- no need to support multipoint links (e.g., polling)

*Error recovery, flow control, data re-ordering all relegated to higher layers!*

# PPP Data Frame

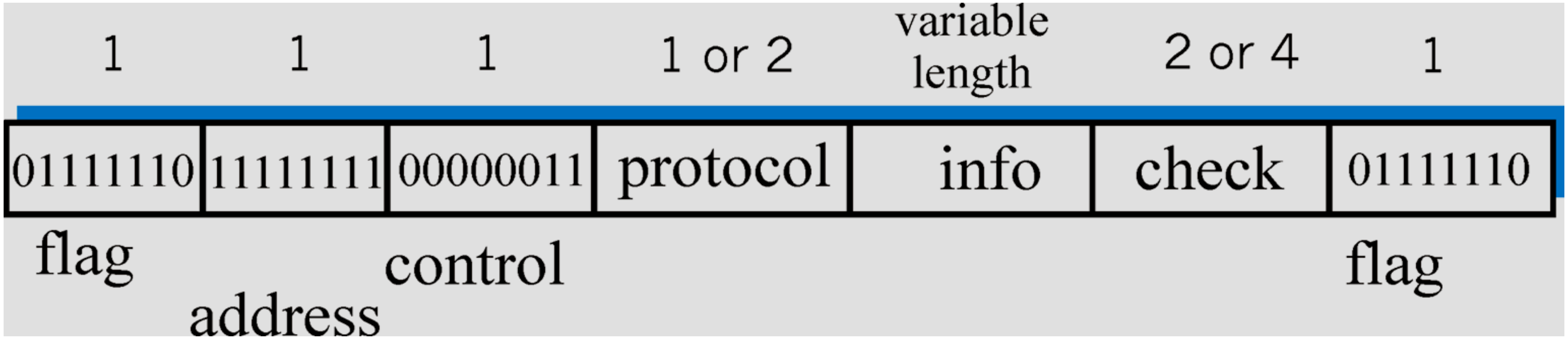
- Flag: delimiter (framing)
- Address: does nothing (only one option)
- Control: does nothing; in the future possible multiple control fields
- Protocol: upper layer protocol to which frame delivered (eg, PPP-LCP, IP, IPCP, etc)



| 1 | 1 | 1 | 1 or 2 | variable length | 2 or 4 | 1 |
|---|---|---|---|---|---|---|
| 01111110 | 11111111 | 00000011 | protocol | info | check | 01111110 |
| flag | address | control | | | | flag |

# PPP Data Frame

- info: upper layer data being carried
- check: cyclic redundancy check for error detection



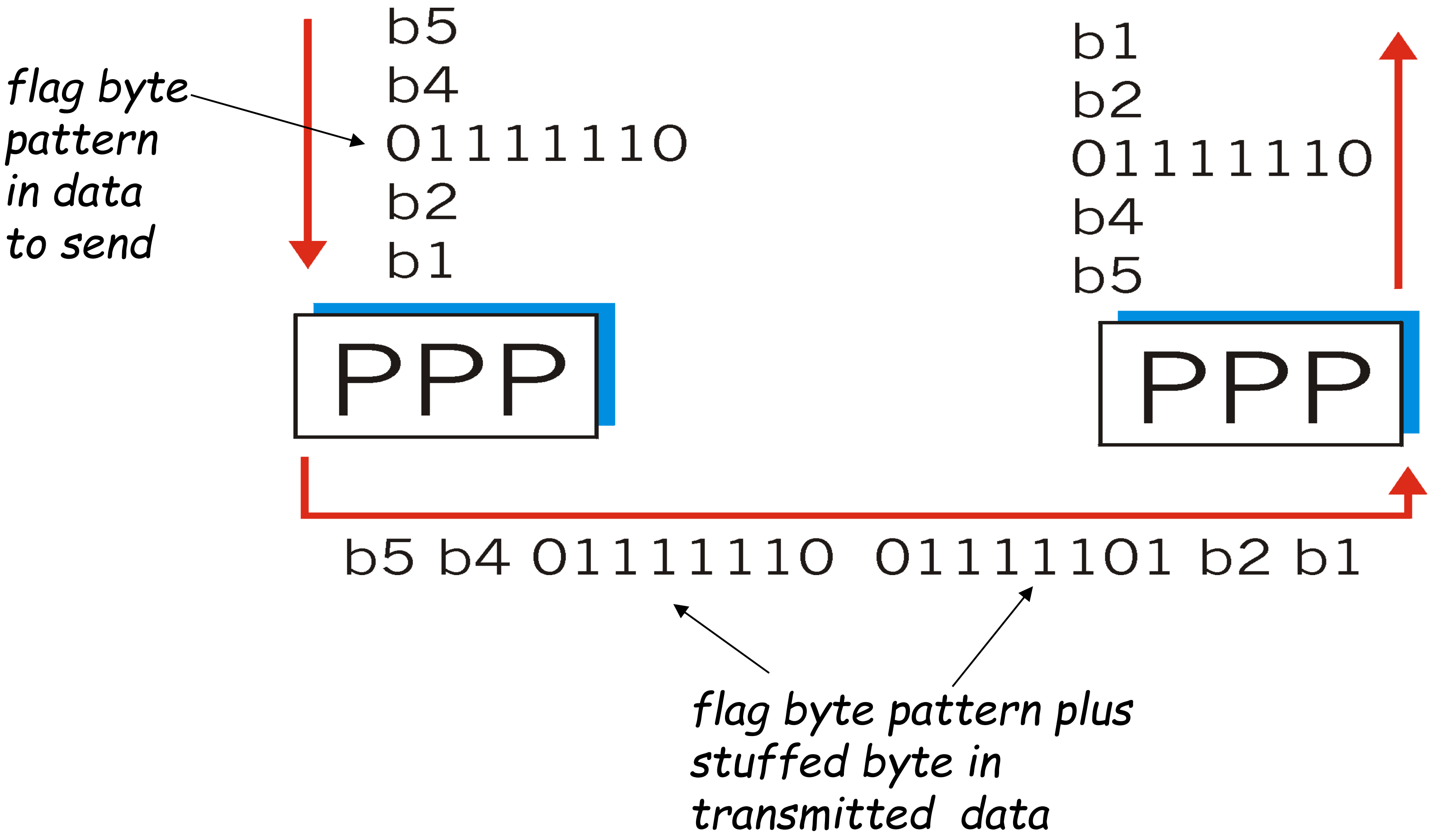
| 1 | 1 | 1 | 1 or 2 | variable length | 2 or 4 | 1 |
|---|---|---|---|---|---|---|
| 01111110 | 11111111 | 00000011 | protocol | info | check | 01111110 |
| flag | address | control | | | | flag |

# Byte Stuffing

- “data transparency” requirement: data field must be allowed to include flag pattern <01111110>
  - Q: is received <01111110> data or flag?

- Sender: adds (“stuffs”) extra < 01111101> byte after each < 01111110> and <01111101> *data* byte
- Receiver:
  - single <01111101> indicates an stuff byte;
  - two 01111101 bytes in a row: discard first byte, continue data reception
  - single 01111110: flag byte

# Byte Stuffing



flag byte pattern in data to send

b5
b4
01111110
b2
b1

PPP

b5 b4 01111110 01111101 b2 b1

flag byte pattern plus stuffed byte in transmitted data

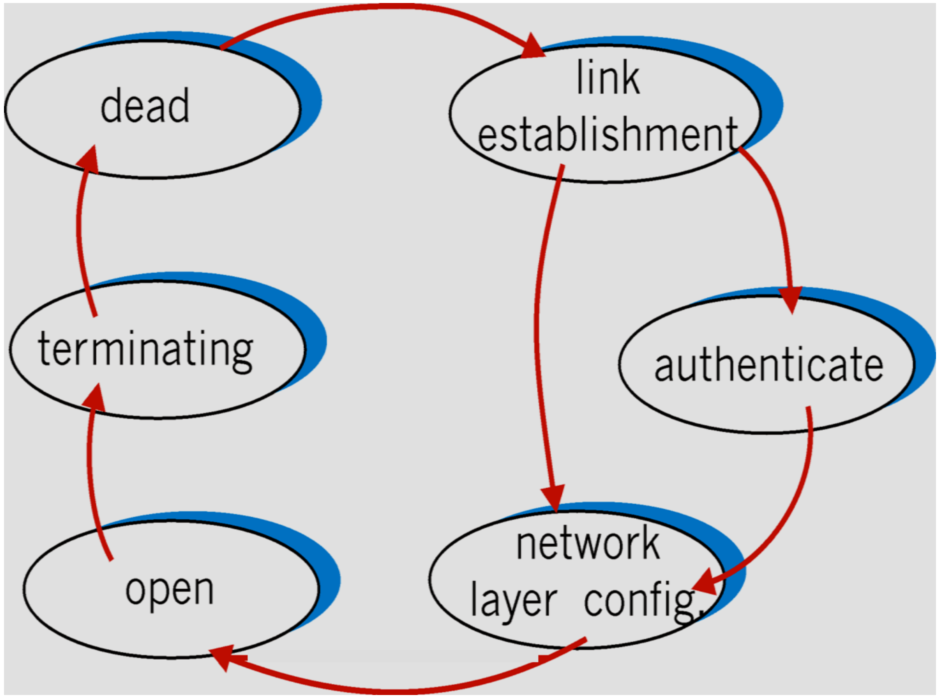
b1
b2
01111110
b4
b5

PPP

# PPP Data Control Protocol

Before exchanging network-layer data, data link peers must

- configure PPP link (max. frame length, authentication)
- learn/configure network layer information
  - for IP: carry IP Control Protocol (IPCP) msgs (protocol field: 8021) to configure/learn IP address



# Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

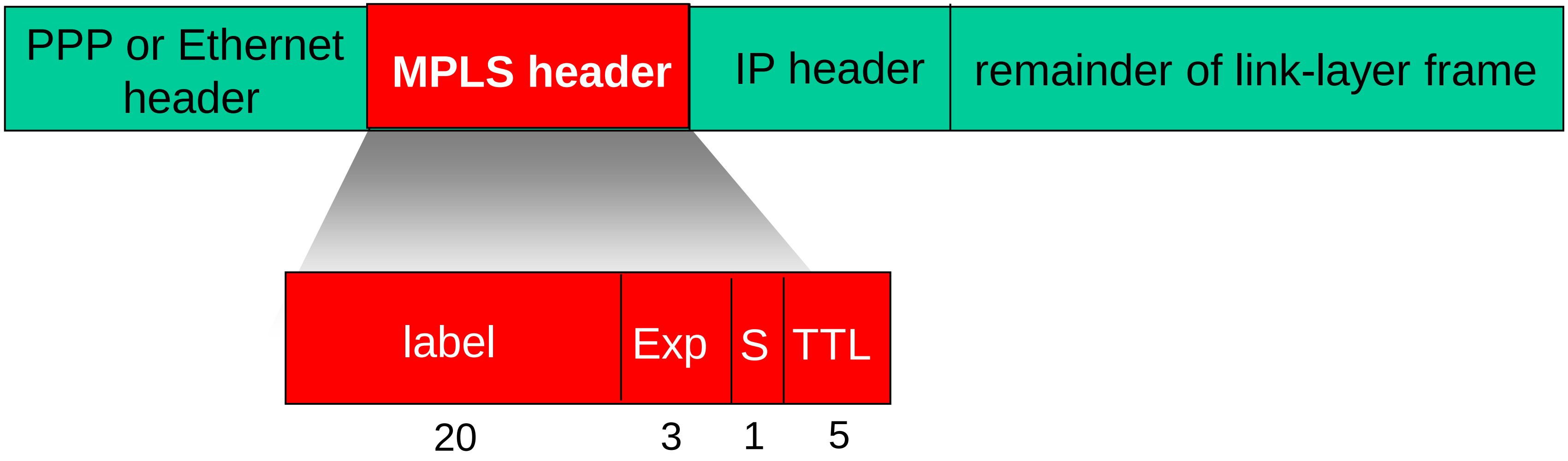
5.5 PPP

5.6 link virtualization: MPLS

5.7 a day in the life of a web request

# Multiprotocol label switching (MPLS)

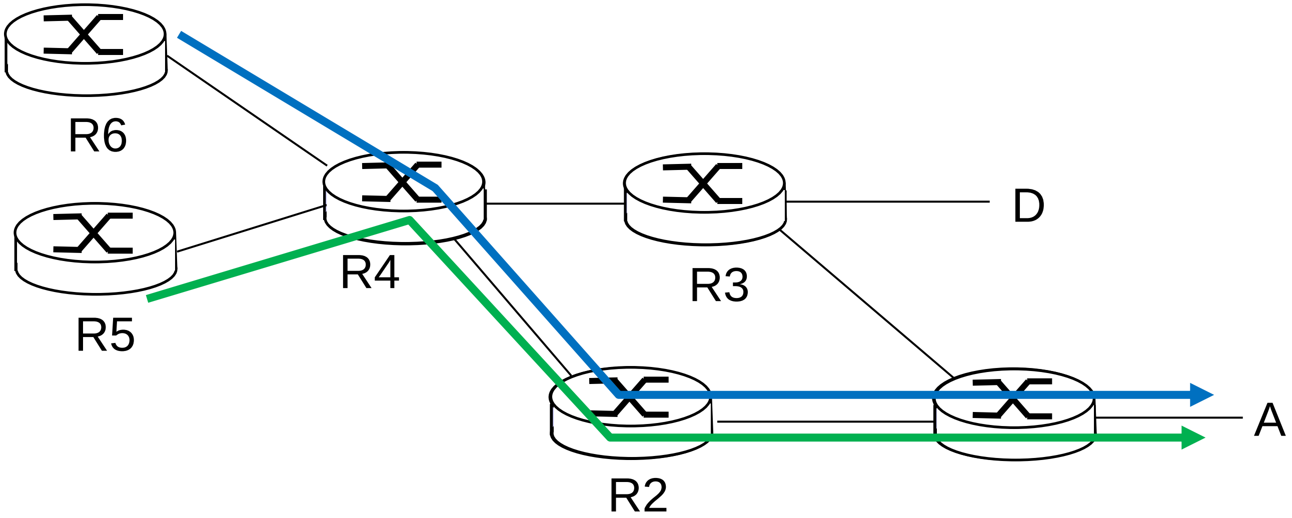
- initial goal: high-speed IP forwarding using fixed length label (instead of IP address)
  - fast lookup using fixed length identifier (rather than shortest prefix matching)
  - borrowing ideas from Virtual Circuit (VC) approach
  - but IP datagram still keeps IP address!



# MPLS capable routers

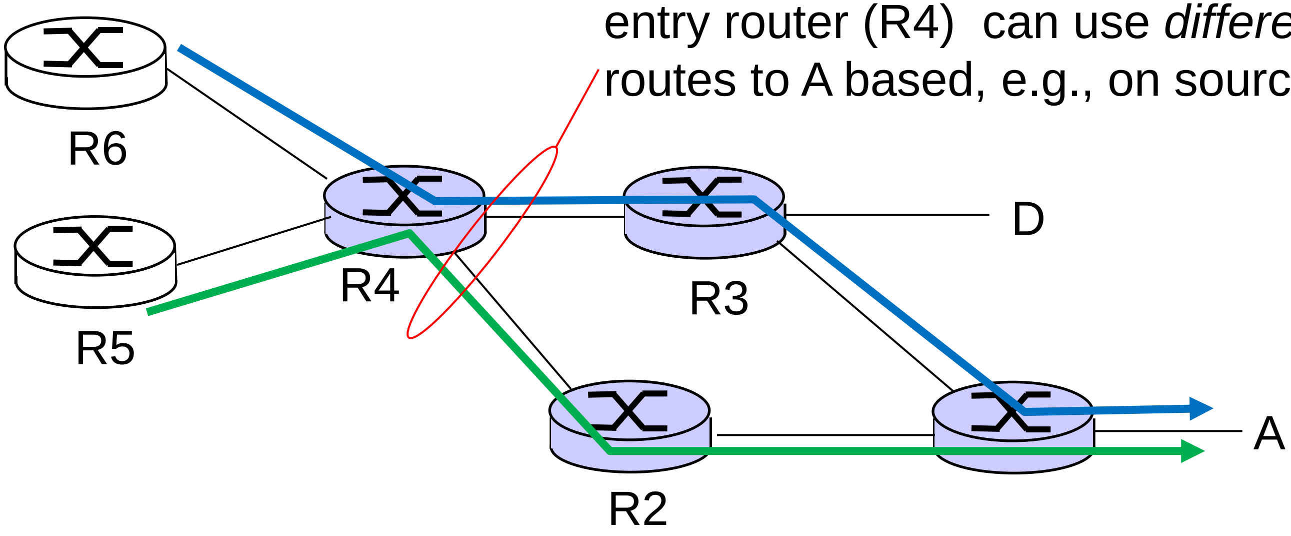
- a.k.a. label-switched router
- forward packets to outgoing interface based only on label value (*don't inspect IP address*)
  - MPLS forwarding table distinct from IP forwarding tables
- *flexibility:* MPLS forwarding decisions can *differ* from those of IP
  - use destination *and* source addresses to route flows to same destination differently (traffic engineering)
  - re-route flows quickly if link fails: pre-computed backup paths (useful for VoIP)

# MPLS versus IP paths



- *IP routing: path to destination determined by destination address alone*

# MPLS versus IP paths



entry router (R4) can use *different* MPLS routes to A based, e.g., on source address

- *IP routing:* *path to destination determined by destination address alone*

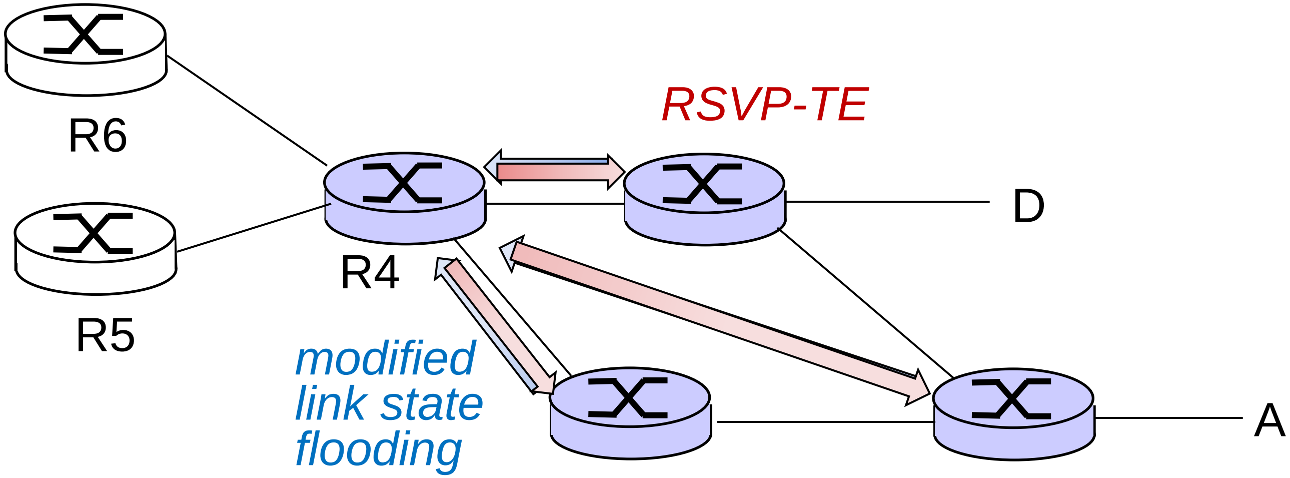
*IP-only router*

- *MPLS routing:* path to destination can be based on source *and* dest. address
  - *fast reroute:* precompute backup routes in case of link failure

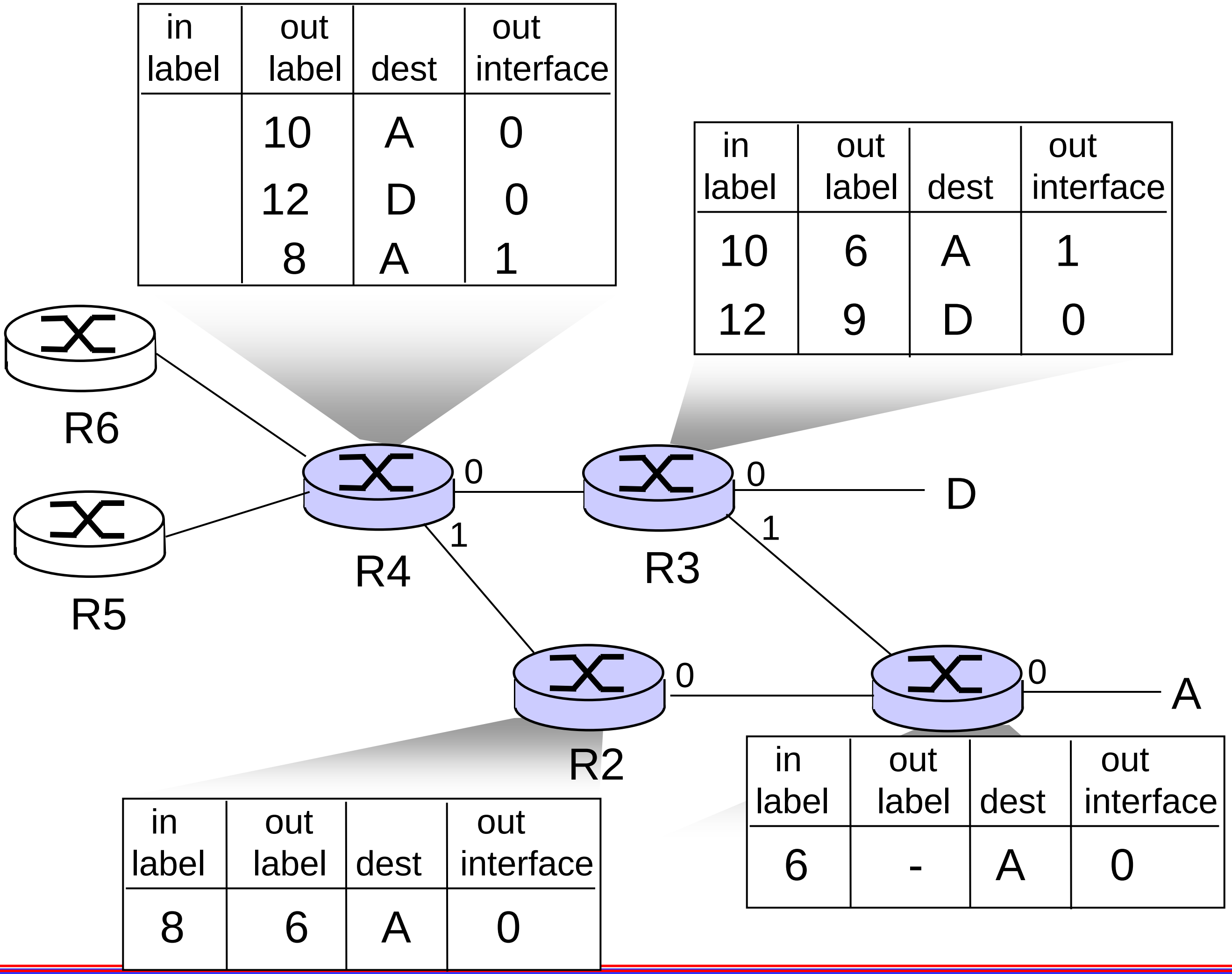
*MPLS and IP router*

# MPLS signaling

- modify OSPF, IS-IS link-state flooding protocols to carry info used by MPLS routing,
  - e.g., link bandwidth, amount of "reserved" link bandwidth
- *entry MPLS router uses RSVP-TE signaling protocol to set up MPLS forwarding at downstream routers*



# MPLS forwarding tables



| in label | out label | dest | out interface |
|---|---|---|---|
| | 10 | A | 0 |
| | 12 | D | 0 |
| | 8 | A | 1 |

| in label | out label | dest | out interface |
|---|---|---|---|
| 10 | 6 | A | 1 |
| 12 | 9 | D | 0 |

| in label | out label | dest | out interface |
|---|---|---|---|
| 8 | 6 | A | 0 |

| in label | out label | dest | out interface |
|---|---|---|---|
| 6 | - | A | 0 |

# Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

5.5 PPP

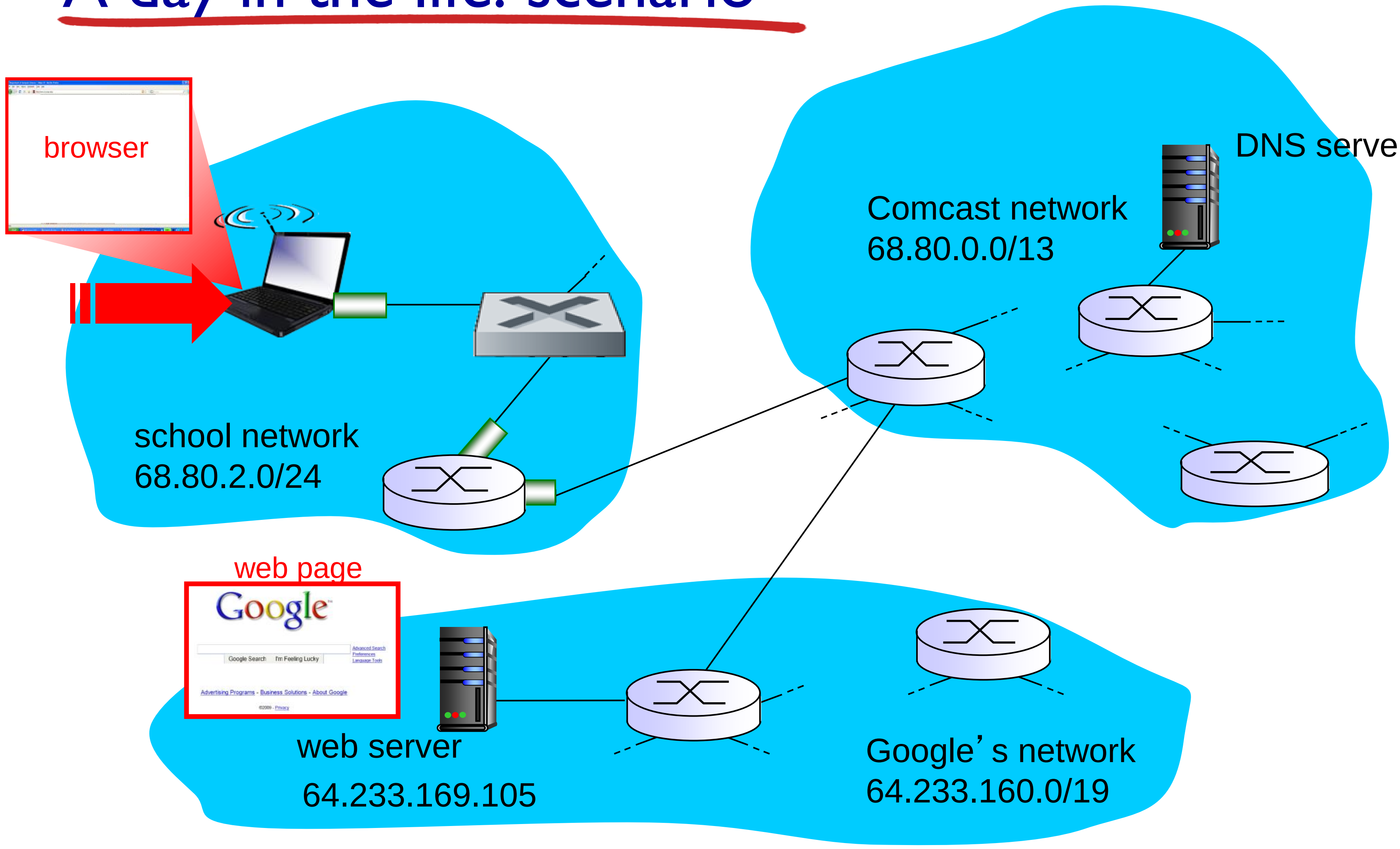
5.6 link virtualization: MPLS

5.7 a day in the life of a web request

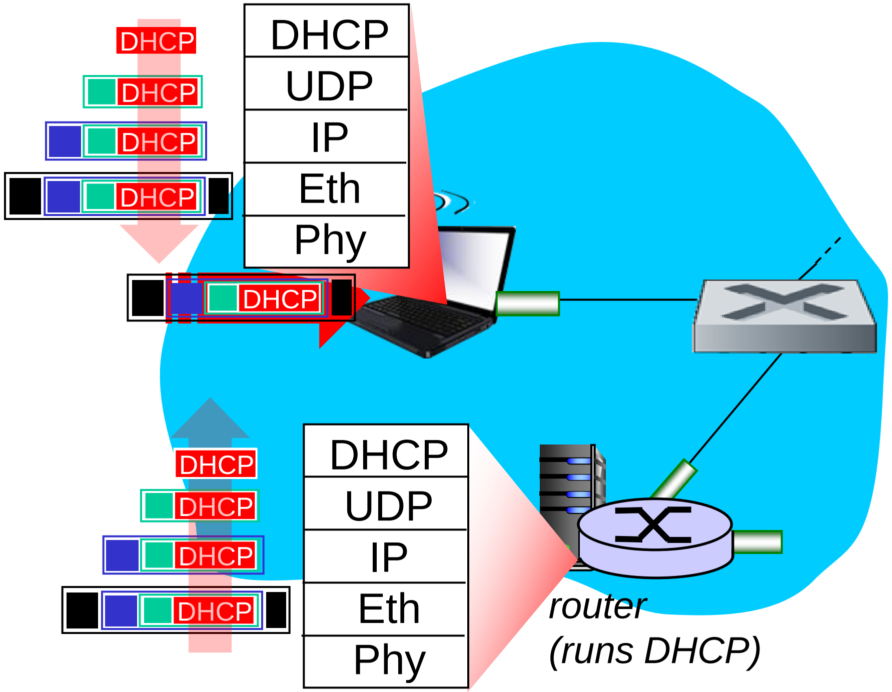
# *Synthesis:* a day in the life of a web request

- journey down protocol stack complete!
  - application, transport, network, link
- putting-it-all-together: synthesis!
  - *goal:* identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
  - *scenario:* student attaches laptop to campus network, requests/receives www.google.com

# A day in the life: scenario

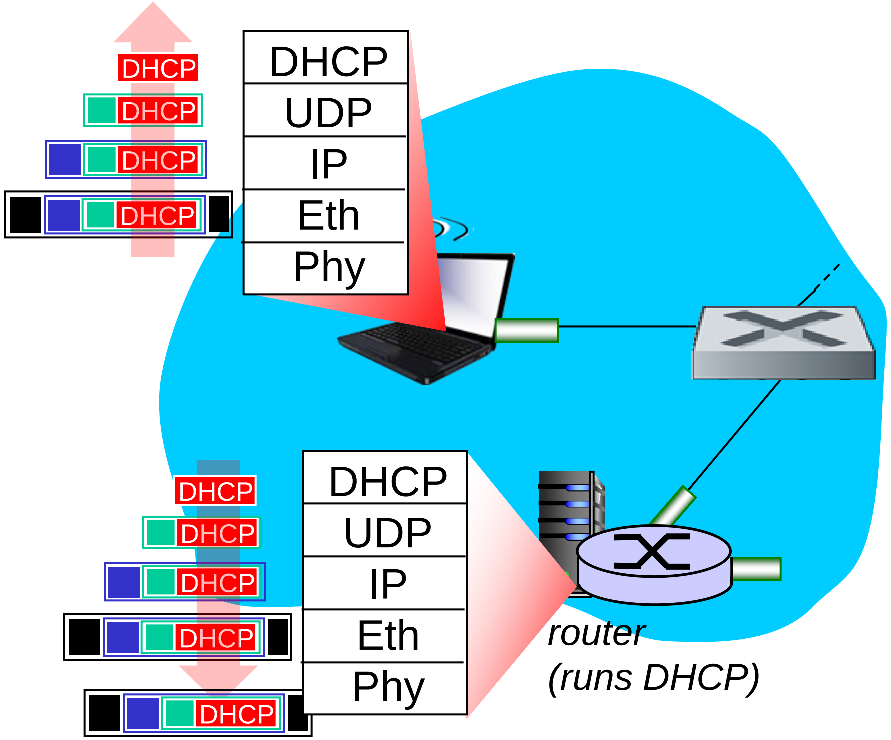


# A day in the life… connecting to the Internet



- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use *DHCP*
- DHCP request *encapsulated* in *UDP*, encapsulated in *IP*, encapsulated in *802.3* Ethernet
- Ethernet frame *broadcast* (dest: FFFFFFFFFFFF) on LAN, received at router running *DHCP* server
- Ethernet *demuxed* to IP demuxed, UDP demuxed to DHCP

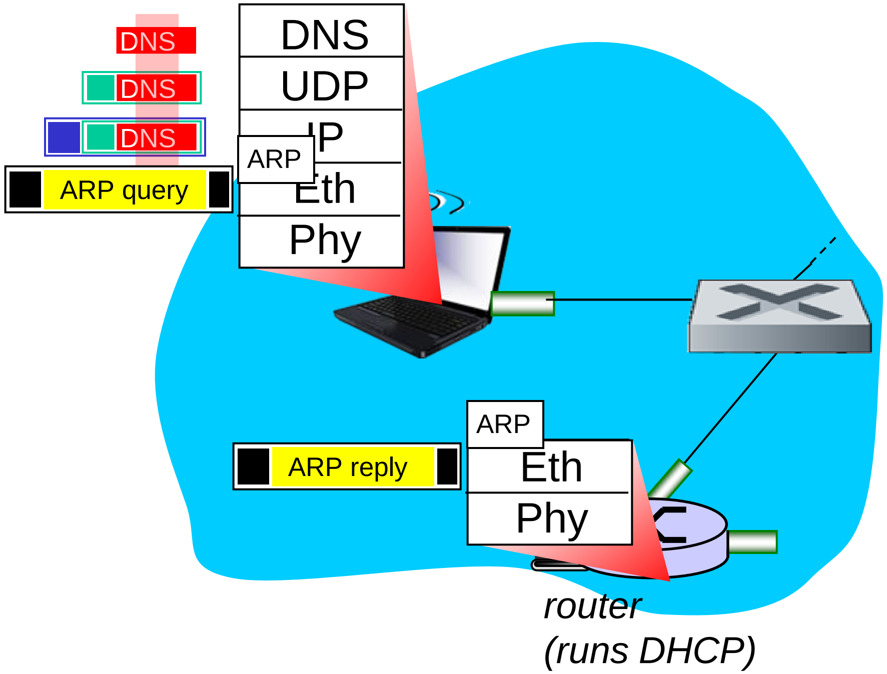
# A day in the life… connecting to the Internet



- DHCP server formulates *DHCP ACK* containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation at DHCP server, frame forwarded (*switch learning*) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply

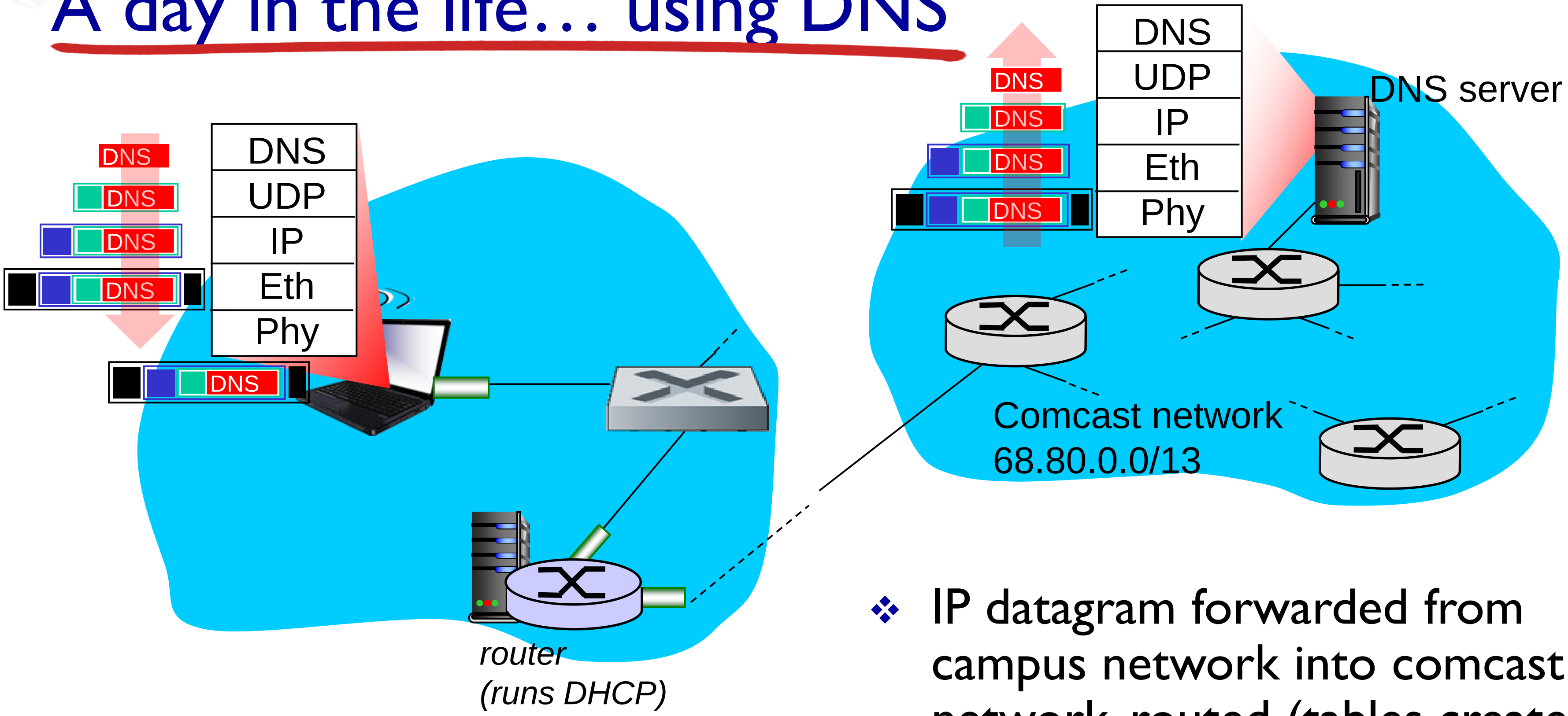
*Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router*

# A day in the life… ARP (before DNS, before HTTP)



- before sending *HTTP* request, need IP address of www.google.com: *DNS*
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: *ARP*
- *ARP query* broadcast, received by router, which replies with *ARP reply* giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

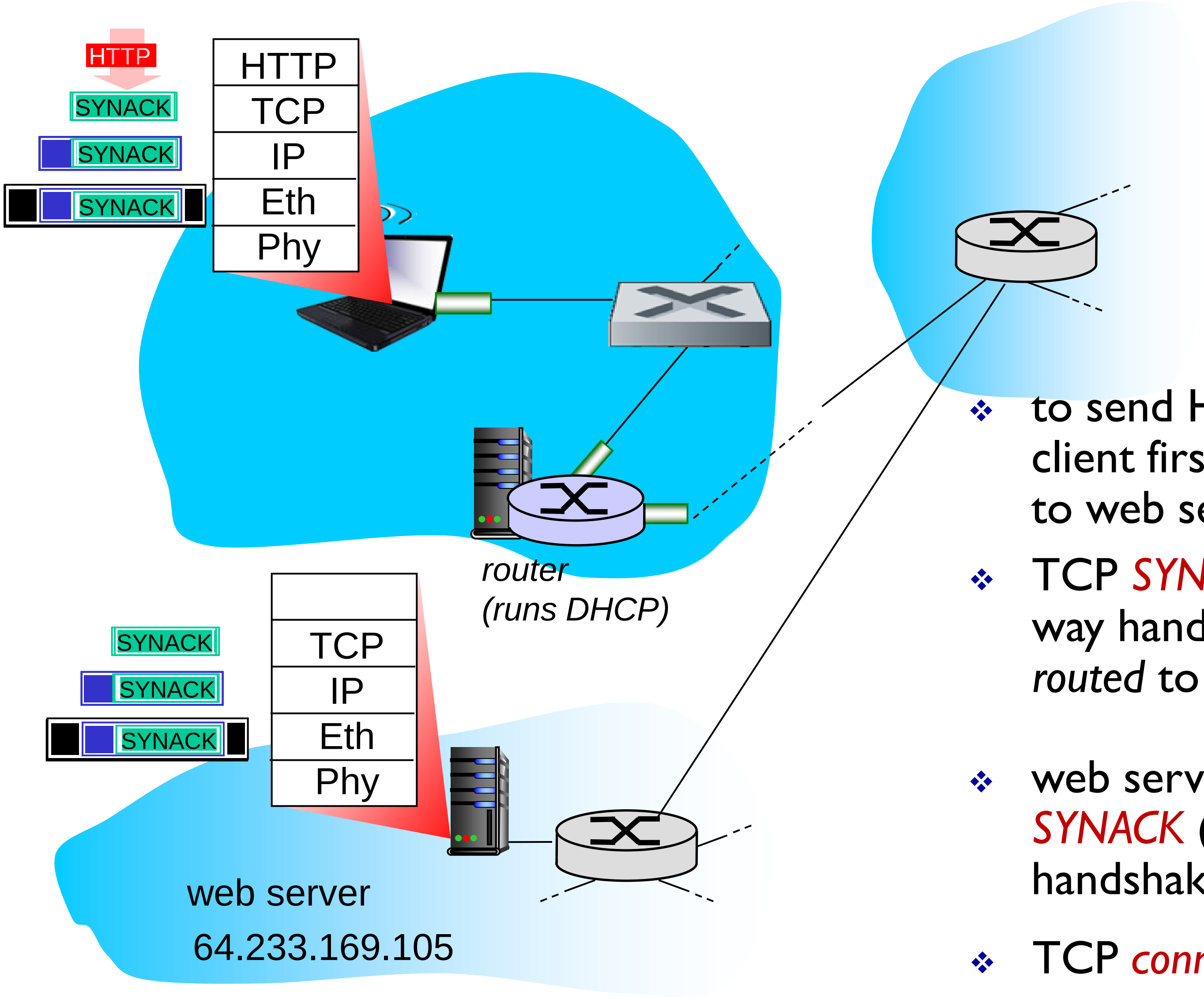
# A day in the life… using DNS



- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router

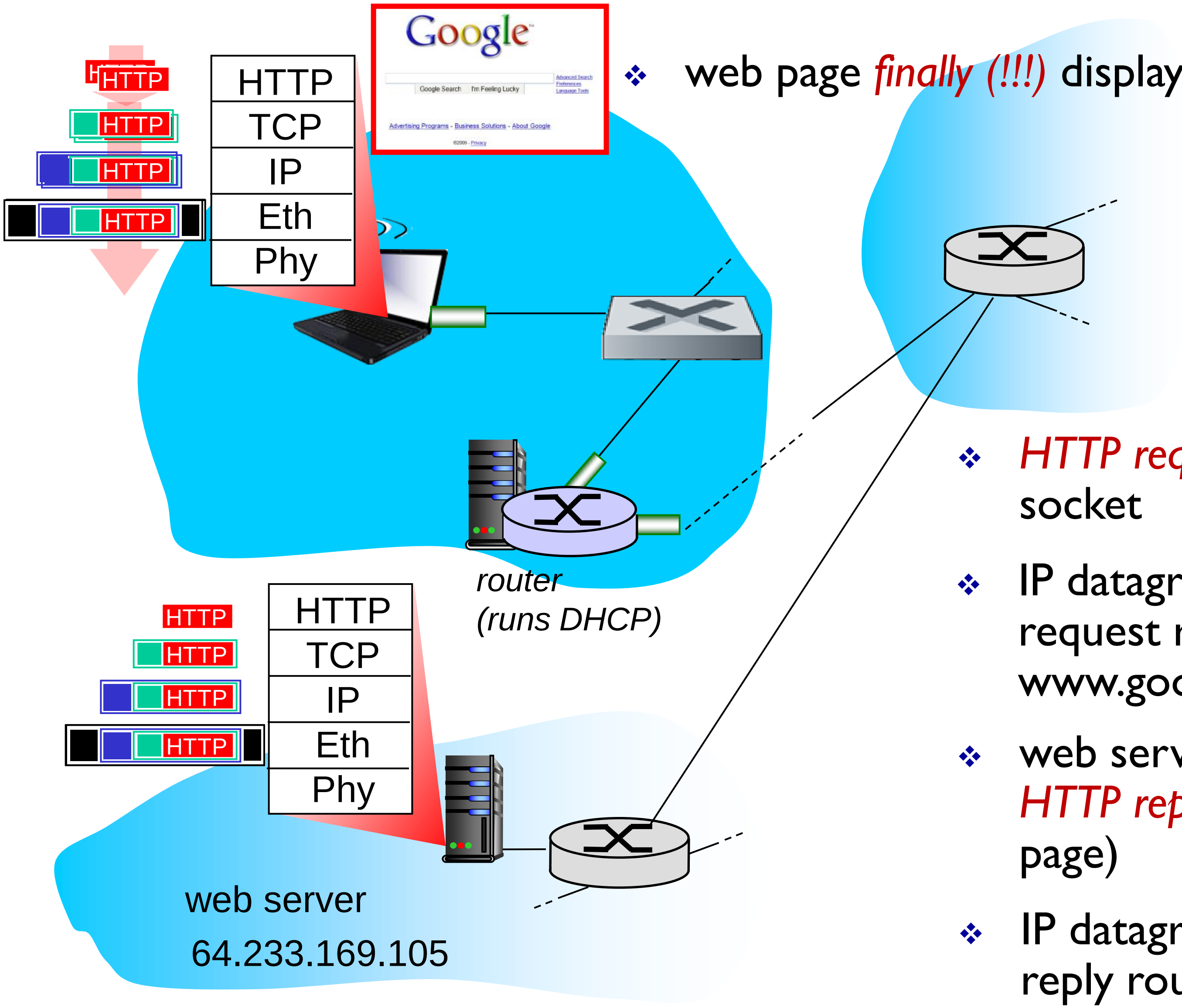
- IP datagram forwarded from campus network into comcast network, routed (tables created by *RIP, OSPF, IS-IS* and/or *BGP* routing protocols) to DNS server
- demux’ed to DNS server
- DNS server replies to client with IP address of www.google.com

# A day in the life…TCP connection carrying HTTP



- to send HTTP request, client first opens *TCP socket* to web server
- TCP *SYN segment* (step 1 in 3-way handshake) *inter-domain routed* to web server
- web server responds with *TCP SYNACK* (step 2 in 3-way handshake)
- TCP *connection established!*

# A day in the life… HTTP request/reply



- web page *finally (!!!)* displayed

- *HTTP request* sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with *HTTP reply* (containing web page)
- IP datagram containing HTTP reply routed back to client

# Chapter 5: Summary

- principles behind data link layer services:
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
- instantiation and implementation of various link layer technologies
  - Ethernet
  - switched LANS, VLANs
  - virtualized networks as a link layer: MPLS
- synthesis: a day in the life of a web request

# Chapter 5: let’s take a breath

- journey down protocol stack *complete* (except PHY)
- solid understanding of networking principles, practice
- ….. could stop here …. but *lots* of interesting topics!
  - wireless
  - multimedia
  - security
  - network management